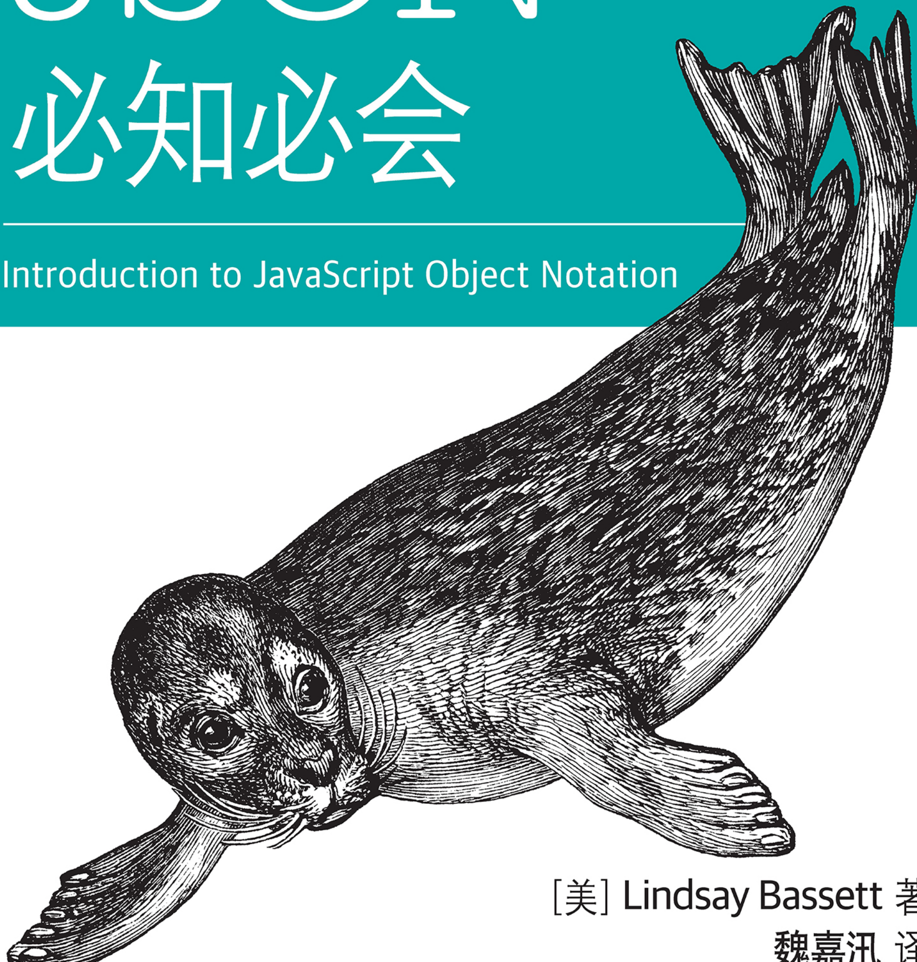
O'REILLY®

TURING 图灵程序设计丛书

JSON 必知必会

Introduction to JavaScript Object Notation



[美] Lindsay Bassett 著
魏嘉汛 译

中国工信出版集团 人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

译者介绍



魏嘉汛

电子科技大学2013级本科生，先后活跃于SysLab工作室、星辰工作室等学生团体。专注于前端开发，希望自己的工作能够服务更多的人。

TURING 图灵程序设计丛书

JSON必知必会

Introduction to JavaScript Object Notation
A To-the-point Guide to JSON

[美] Lindsay Bassett 著
魏嘉汛 译

人民邮电出版社
北京

图书在版编目（ＣＩＰ）数据

JSON必知必会 / （美） 巴塞特 （Bassett,L.） 著 ；
魏嘉汛译. -- 北京 ： 人民邮电出版社， 2016.5
（图灵程序设计丛书）
ISBN 978-7-115-42207-1

Ⅰ. ①J… Ⅱ. ①巴… ②魏… Ⅲ. ①JAVA语言－程序
设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2016)第083118号

内 容 提 要

越来越多的 IT 从业者需要学习或了解 JSON。本书即针对这一现状，围绕 JSON 这一主题的核心展开讲解，首先介绍 JSON 语法、语法验证、数据类型、模式验证、安全问题，再讲解 JSON 作为数据交换格式所扮演的种种角色，还涉及 jQuery、AngularJS 以及 CouchDB 等技术的进阶介绍，并给出了大量代码示例，是一本让读者快速透彻地了解 JSON 的指南。

本书适合有一定基础的 Web 开发人员和各类语言开发者阅读。

◆ 著　　[美] Lindsay Bassett
　译　　魏嘉汛
　责任编辑　岳新欣
　执行编辑　张　曼
　责任编辑　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
　邮编　100164　　电子邮件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京　　　　　印刷

◆ 开本：880×1230　1/32
　印张：4
　字数：115千字　　　　2016年 5 月第 1 版
　印数：1－3 000册　　　2016年 5 月北京第 1次印刷
著作权合同登记号　图字：01-2016-2264号

定价：35.00元
读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广字第 8052 号

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会聚集了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——Wired

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——CRN

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

目录

前言

从事 Web 开发这么多年，我明白了一件事情：无论我投入多少精力学习这方面的知识，都很难抽时间来总结这些知识。这个迅猛发展的技术世界根本不会在意你有多忙。它不会说："放轻松点，我会给你时间学习的，我明白由于家庭的原因，你很难抽出时间安静地学习。"它只会说："快跟上，否则你就会落后。"

在我写作本书时，这样的思绪就一直萦绕在脑海中。这是一本很薄的书——我在书中刻意回避了"JSON 的历史"之类的话题。

尽管我对 JSON 之父 Douglas Crockford 心存感激，但我在书中没有提到他，也没有提到 JSON 是怎么一步步成长为今天的样子的。本书仅专注于 JSON 今天的样子。如果你想要了解 JSON 的历史，维基百科会是一个好去处（https://en.wikipedia.org/wiki/JSON#History）。

本书围绕 JSON 这一主题的核心展开讲解，力求抓住重点，深入浅出。毕竟，我们 IT 从业者都很忙。

读者对象

由于本书是为那些忙碌的 IT 从业者准备的，所以对读者肯定有一定的要求。你可能是刚入门的前端开发人员，或是在服务端应用开发中浸淫多年的老开发者，需要学习一些与 JSON 相关的知识来构建 Web API。你也可

能是 PHP、Ruby、C、Java 或 ASP.NET 等语言的开发者，等等。越来越多不同行业的 IT 从业者都想要或需要学习 JSON。

本书中，我会避免使用过多的专业术语，并会为 Web 开发的新入行者解释一些基本概念。尽管我希望我的书适合每一位读者，但我肯定会假设你有一些相关的基础知识。如果你是昨天才进了 Web 开发的门，那么这本书可能不是你的最佳首选。

我假设你对以下概念有所了解。

- HTML
  你需要了解 HTML 的用途，能够辨析其结构，并且至少认识一小部分 HTML 文档的标签。

- JavaScript
  你需要了解 JavaScript 的用途，认识 `<script>` 标签，理解函数、变量之类的概念。如果你是 JavaScript 新手，不用担心，我会尽可能简化代码示例。

- 编程概念
  对于新手，我解释了一些基本概念，如对象、数组。然而，如果你还没使用过任何一种编程语言，那么本书可能不适合你。

本书结构

多年以来，我一直在不断地接触并学习新技术，学习过程中也常常伴有项目进度的压力。所以在购买新的技术书之后，我会快速浏览其中的教程，尝试着尽快获取足够的信息以理解我现在所学的东西。我会带着以下三个基本问题来快速浏览一本书。

- 它是什么？
- 我可以用它做什么？
- 那些别有用心的人会用它做什么？

在写作本书时，我针对这些问题给出了直切要害的答案，因此你不必费神搜寻这些答案。

在本书的第 1~4 章，我们将学习一些关于 JSON 的基础知识。首先，我会告诉你“它是什么”。以此为出发点，我们会一步步了解它的语法、语法验证、数据类型与模式验证。

第 5 章专注于研究 JSON 中的安全问题。我会在这里介绍客户端和服务端的概念，这一对概念在本书余下的章节中至关重要。同时，这一章也回答了第三个问题——“那些别有用心的人会用它做什么”。

剩余的章节会详细介绍 JSON 作为数据交换格式所扮演的种种角色。这些章节会为你解答第二个问题，即“我可以用它做什么”。

这些章节包含大量的 JSON 示例，以及使用 JSON 的交互技术。下面列举一些关于第 6~9 章的重要内容。

- 技术

  这部分重点介绍了 jQuery、AngularJS 以及 CouchDB 等技术的进阶知识。把其中的每一个技术主题单拿出来都足够写一本书了（而且市面上确实也有这类书籍）。我没有介绍如何安装它们，也没有进行深入的讲解。我重点讲解的是如何将它们与 JSON 相结合。

  如果你希望尝试我在书中给出的关于这些技术的示例，那你可能需要做些工作来把环境配置起来。不过，书中的这些例子还是很简单的。如果你已经把环境搭建好并且基本可以运行的话，那么就可以着手尝试这些例子了。

- 代码示例

  本书包含大量代码示例，其中部分示例所用的编程语言你可能没有接触过。虽然我不会讲解这些语言的语法，但你也不必因为不清楚语法而感到惊慌。对于你来说，最重要的是理解这段代码做了什么事情，关于这方面我会为你提供足够的解释。

  本书中的所有代码示例都可以在相应的 GitHub 代码库（https://github.com/lindsaybassett/json）中获取。

编写第 6~9 章的最终目的是让你明白 JSON 是如何在我们的实际生活中使用的，你可以尝试着把它们用在你自己的项目之中。如果你从没见过把

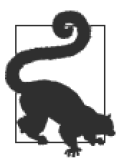
JSON 作为文档用在文档存储数据库中，那么你会在项目中使用它吗？知识就是力量。

本书的每一章都尽可能在“内容表述应简洁明了”与“要传达足够多的信息”之间寻求平衡，以免遗漏重要知识点。编写这本书的意图是让你既能快速理解并使用 JSON，又能深入理解 JSON 的本质及用途。

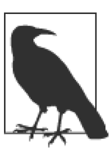
排版约定

本书使用的排版约定如下所示。

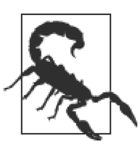
- 楷体
  表示新术语。
- 等宽字体（`constant width`）
  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键字等。
- 等宽粗体（**`constant width bold`**）
  表示应该由用户输入的命令或其他文本。



该图标表示提示或建议。



该图标表示一般注记。



该图标表示警告或警示。

使用代码示例

补充材料（代码示例、练习等）可以从 https://github.com/lindsaybassett/json 下载。

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序或文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可，销售或分发 O'Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无需获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN。比如："*Introduction to JavaScript Object Notation* by Lindsay Bassett (O'Reilly). Copyright 2015 Lindsay Bassett, 978-1-491-92948-3."

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

Safari® Books Online

Safari Books Online（http://www.safaribooksonline.com）是应运而生的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。技术专家、软件开发人员、Web 设计师、商务人士和创意专家等，在开展调研、解决问题、学习和认证培训时，都将 Safari Books Online 视作获取资料的首选渠道。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、

FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

O'Reilly 的每一本书都有专属网页，你可以在那儿找到本书的相关信息，包括勘误表、示例代码以及其他信息。本书的网站地址是：

http://shop.oreilly.com/product/0636920041597.do

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：http://facebook.com/oreilly

请关注我们的 Twitter 动态：http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：http://www.youtube.com/oreillymedia

致谢

首先，我要感谢我的丈夫 Rhett，感谢他给予我的鼓励与支持，以及对我整日闭门写书的谅解。

同样我要感谢 Douglas Crockford，他发明了 JSON，我才得以写出一些有趣的东西。我还要感谢我的技术审稿人 Shelley Powers 和 Tom Marrs，他们给我的意见极具建设性，让本书增色不少。当然，书中的一切错误，责任都在我。

感谢我的编辑 Meg Foley，能和 Meg 一起工作真的很棒。

最后我要感谢 O’Reilly 给了我的书一个家，并将它出版发行。我一直都是 O’Reilly 图书的粉丝，而和他们共事之后，我甚至变成了“骨灰粉”。

第 1 章

什么是JSON

在深入讨论 JSON 之前，先让我们对它有一个感性的认识。宏观上看，JSON 是一种轻量的数据格式，在各种地方传递数据。如果单用眼睛看，JSON 里的数据是被保存在花括号（{}）中的，而如果从用途上进一步分析，最终我们会得出结论：JSON 是一种数据交换格式。

1.1 JSON是一种数据交换格式

数据交换格式是一种在不同平台间传递数据的文本格式。除 JSON 外，你也可能听说过 XML 这种数据交换格式。像 XML 和 JSON 这样的数据交换格式非常重要，我们需要它们来实现不同系统间的数据交换。

举个例子，假如有这样一个世界，它由数百个散布在海洋中的小岛所组成。每个海岛都是相互独立的，并有自己独特的语言和习俗。这些岛上都有许多商人，他们需要在海岛间进行长途航行。这种对外贸易是所有海岛经济必需的组成部分，也有助于提高岛民的生活水平。而这一切的实现都要归功于那些训练有素的送信海鸥。

这些海鸥在岛间飞行，携带着需求量最大的货物的信息。商人根据这些信息来决定他们的下一站，以及在长途航行前应储备哪些货物。也正是凭借这些关键的数据，各个海岛间才可以互通有无，共同繁荣。

别忘了，每个海岛的语言都不同。如果这些信息用各种不同的语言编写，那么每个海岛都要花上一大笔钱来研究各种语言，并组建一支翻译团队。这既昂贵，又费时。不过岛民们十分聪明，他们决定统一使用一种语言，用一种标准的数据格式来传达贸易数据。这样，每个海岛都只需雇用一个懂得这一数据格式的翻译就好了，由他们来解读海鸥带来的贸易报告。

这个海岛的例子其实就映射出了我们在实际生活中所使用的技术。我们的生活中充满了各种系统，它们所使用的语言和架构都不尽相同。而对于使用这些系统的企业和组织来说，它们之间相互通信的能力又是不可或缺的。但如果每一个系统都必须有一个专门针对其他所有系统的数据组织形式的翻译组件，那么它们之间的交流便要消耗许多时间和资源，这显然是不合理的。所以，这些系统间也需要一种单一的数据格式，以及单一的翻译组件。

JSON 就是这样一种被许多系统用于交换数据的数据交换格式。有人把它叫作“数据交换格式”，甚至直接叫“数据格式”。在本书中，我们把 JSON 看作一种数据交换格式，是因为“交换”往往意味着两个或多个实体之间的相互交流。

然而，不是所有的系统都支持使用 JSON 来交换数据。数据交换格式有很多，如前面提到的 XML（extensible markup language，可扩展性标记语言），可能早在 JSON 被发明前就已经在应用了。毕竟现实世界不会像例子中的海岛世界那么简单。有许多系统可以并还在使用像 XML 这样的格式，或是用表格和分隔符来表示数据的格式，如逗号分隔值（CSV）。现实中的每个“海岛”所选择的数据交换格式，也通常会和数据格式与“海岛”的风俗、语言、建筑结构等因素间的联系相挂钩。

示例中的海岛世界里，每一个海岛都有它自己的语言。而海鸥所传送的报告上的数据所用的格式，是一种与语言无关的格式。这使得每个岛只需要雇用一个解释贸易报告的翻译即可。JSON 也一样，只不过数据不是通过海鸥传送的，而是通过网络中的 0 和 1 这样的信号传送。翻译自然也不是人，而是系统的一个解析器，用于将数据转换为系统可以读取的形式。

1.2 JSON独立于编程语言

JSON 的全称是 JavaScript Object Notation（JavaScript 对象表示法）。这个名字可能会让人误以为要想理解和使用 JSON，得先学习 JavaScript。诚然，在学习 JSON 前学一点 JavaScript 肯定会有帮助，毕竟 JSON 源于 JavaScript 的一个子集。但如果你以后用不到 JavaScript，那也没有必要去学习它，因为数据交换格式是独立于语言的。你仍可以在你自己的系统中使用你自己的语言。

JSON 基于 JavaScript 对象字面量。关于“如何”使用 JSON，更适合在关于句法（第 2 章）和数据类型（第 3 章）的章节讨论。这一章中最重要的还是讨论“为什么”。既然数据交换格式需要独立于语言，那么 JSON 不仅源于一门语言，还在名字中给这种语言打广告，这是否有些自相矛盾？为什么？

回到海岛的例子，想象一下众海岛代表开会决定数据格式时是怎样一种情形。当数以百计的海岛代表们前来参会，并尝试创造一种数据格式的时候，他们要做的第一件事就是寻找共同点。

每一个海岛的语言也许都是独一无二的，但是海岛居民们也会在其中发现共同点。绝大多数语言都由人来发声，并包含一种由某种字符组成的书面语言形式。此外，大多数语言还都有面部表情和手势。也许有一些海岛的居民使用其他的方式来沟通，如敲击木棒或是眨眼睛，不过大多数海岛都会在其语言的口语形式和书面形式中找出相同点。

现实世界中同样有着数以百计的编程语言。虽然有一些非常流行且广泛使用的语言，但是语言大家族整体是多样化的。当大学生们选择计算机科学专业，为编程生涯做准备时，他们不会学习所有的语言。他们一般会从一门语言开始学习，而且语言的选择并没有学习编程中通用的概念重要。一旦他们掌握了这些概念，学习另一门编程语言时就会轻松很多，因为他们已经有能力分辨那些共有的特征和功能。

再来看看 JSON，如果我们将“JavaScript 对象表示法”中的“JavaScript”去掉，剩下的就是“对象表示法”。所以忘了 JavaScript 吧。可以说，我们使用的是一种基于对象表示法的数据交换格式。而“对象”在编程中，尤

共是在面向对象编程（OOP）中，是很常见的概念。绝大多数计算机科学专业的学生都会在学习编程时接触到对象的概念。

我们不过多地解释对象，而是把重点放到“表示法”上。*表示法*是指一个可以表示诸如数字或单词等数据的字符系统。不管你是否理解什么是对象，都不难看出用某个符号去描述编程语言中共有概念的价值所在。

再回到海岛的例子，岛民们发现了一种出现在大多数语言中的表示法。举例来说，许多岛民都使用计数算筹来表示数字，具体用法也大多类似；同时，他们也都能理解一些表示生活中常见事物（如小麦或布）的符号。甚至那些用眨眼来交流的岛民都能接受这种格式。

尽管大部分海岛之间达成了共识，仍有一小部分海岛，比如那个通过敲击木棒来交流的海岛，发现他们不能理解这种格式。一种好的数据交换格式必然要适用于大多数系统，但也常常会有小部分不适用的系统。这时我们常会用到一个术语：*可移植性*。可移植性，或者说在平台和系统间传输信息的兼容性，是一种数据交换格式所追求的一个重要指标。

回到表示法这一概念上，虽说 JSON 这一表示法起源于 JavaScript，但是真正重要的是表示法本身。JSON 不仅独立于语言，而且使用了一种在许多编程语言中都能找到共同元素的表达方式。通过这种表达数据的方式，即便是那些不支持面向对象编程的语言，也会发现 JSON 这种格式是可以接受的。

1.3　专业术语和概念

本章涵盖了以下专业术语。

- JSON
  JavaScript 对象表示法（JavaScript Object Notation）。
- *表示法*
  一个用于表示诸如数字或单词等数据的字符系统。
- *数据交换格式*
  用于在不同的平台或系统间交换数据的文本。

* 可移植性

  以一种对双方系统都兼容的方式在平台间传递信息。

我们还讨论了以下重要概念。

* JSON 是一种数据交换格式。
* JSON 独立于编程语言（你不必先学习 JavaScript）。
* JSON 基于 JavaScript 对象字面量表示法（重点在于“表示法”）。
* JSON 表达数据的方式对通用的编程概念都很友好。

第2章

JSON语法

2.1 JSON基于JavaScript对象字面量

在英语中，“literal”（字面上的）是一个形容词，用来表示所说的话所表达的是其字面意思，不是隐喻。比如你的朋友对你说：“她不知从哪儿突然冒了出来，吓得我三明治都掉了。”他所说的“三明治掉了”就是字面意思，而不是隐喻。

而编程语言中的 literal（字面量）是一个名词。所谓*字面量*，是对数据值的具体表示。它的字面意思与其想要表达的意思是完全一致的。如果你对编程相关的概念不熟悉的话，可能会觉得有些奇怪。下面让我们快速了解字面量。

结账时你习惯用现金还是刷卡？假如我现在在三明治店准备结账，当我给收银员 5 美元现金时，我是眼睁睁看着钱从我的钱包“飞”走的；而当我刷卡结账时，虽然我没有亲眼看到，但是我也知道我的账户里少了 5 美元。

而在编程中，我们经常用变量来代表值。例如，我会在表达式中这样使用一个称为 x 的*变量*：

```
x = 5
```

然后，我可能想让 *x* 加 5：

```
x = x + 5
```

现在，虽然我们没有看到 10，但我们知道 *x* 的值变成 10 了。在这个示例中，*x* 就是变量，5 就是字面量。在前面举的三明治店的例子中，我们可以说 5 美元是字面量，信用卡是变量。我们看到的实际值，就是字面的值。

在"*x* = 5"这个例子中，5 是一个数字字面量。数字是一种数据类型。还有一些其他的数据类型，如字符串（由字符组成）、布尔值（真或假）、null（空）、值的集合，以及对象。我们使用数字来直观地表示一个数值。同样，我们也会用 `true/false` 或 `0/1` 来直观地表示布尔值。如果你熟悉对象的概念，那么你会知道对象的表示并不是一件容易或简单的事。不过放心，即使你对这个概念不熟悉，也不影响接下来的阅读。

编程中对象的概念和现实生活中的对象（比如你的鞋子）很相似。你可以用一些特征或属性，比如颜色、风格、品牌等，来描述你的鞋子。有些属性值可能是一个数字，如鞋号，有些可能是布尔值（真 / 假），比如"有蕾丝花边"。详见示例 2-1。

示例 2-1：使用 JSON 描述我现在穿的鞋子

```
{
    "brand": "Crocs",
    "color": "pink",
    "size": 9,
    "hasLaces": false
}
```

现在先不要关注鞋子示例的语法，本章稍后会详细说明。示例想表达的是，程序（甚至是人）可以从字面上读出我的鞋子的一些属性。如果将这个用 JSON 来描述鞋子的示例视为字面量，其数据类型就是对象。我们发现，这种对象的字面量将属性用一种我们直接可见（且可读）的方式展现出来。我们将这一对象的特征或属性所采用的表示方法称为名称 – 值对。

JSON 是基于 JavaScript 对象字面量的。注意是"基于"。在 JavaScript（以及大多数包含对象概念的编程语言）中，对象里面常常包含函数。因此我不仅能直接使用 JavaScript 对象来表示鞋子的属性，还能创建一个叫

“walk”的函数。

而且，数据交换格式的核心是数据，所以 JSON 中并不会涉及 JavaScript 对象字面量中的函数。JSON 所基于的 JavaScript 对象字面量单纯指对象字面量及其属性的语法表示。这种属性表示方法也就是通过名称 – 值对来实现的。

2.2 名称–值对

在计算机界，名称 – 值对的概念非常流行。它们也有别的名字，像键 – 值对、属性 – 值对或字段 – 值对等。本书中，我们将使用名称 – 值对这个称呼。

如果你对名称 – 值对这一概念已经很熟悉了，那么 JSON 看上去也会很亲切。当然，不熟悉也没有关系。让我们来简单地了解一下名称 – 值对。

在名称 – 值对中，你首先要声明一个名称，例如 `"animal"`。然后把它凑成一对：一个名称加一个值。我们来给这个名称（本例中的 `"animal"`）一个值。为简单起见，我们给它一个字符串类型的值。在 JSON 中，名称 – 值对的值还可以是数字、布尔值、null、数组或对象。我们会在第 3 章对除字符串外的其他类型的值进行深入讨论。现在，我们给这个名称为 `"animal"` 的名称 – 值对加上一个字符串类型的值 `"cat"`：

```
"animal" : "cat"
```

`"animal"` 就是名称，`"cat"` 就是值。分隔名称和值的方式有很多。比如，现在我给你一份公司的雇员名录，其中包含他们的头衔，那这个列表很可能看上去像下面这样：

- Bob Barker，CEO
- Janet Jackson，COO
- Mr. Ed，CFO

以上列表使用逗号来分隔员工头衔（名称）和姓名（值）。此外，我把值放在了左边，把名称放在了右边。

JSON 中使用冒号（:）来分隔名称和值。名称始终在左侧，值始终在右侧。让我们再看一些例子：

```
"animal" : "horse"

"animal" : "dog"
```

很简单吧？一个名称加一个值，这就是名称－值对。

2.3 正确的JSON语法

现在让我们来了解一下JSON的语法。名称，也就是我们示例中的"animal"，始终需要被双引号包裹。双引号中的名称可以是任何有效的字符串，所以你的名称即使看起来像下面这样，在JSON中也是完全合法的：

```
"My animal": "cat"
```

你甚至可以在名称中使用单引号：

```
"Lindsay's animal": "cat"
```

现在你知道这是合法的JSON了，但我还要再跟你多说一句：最好不要这么做。这是因为，JSON中的名称－值对是一种对许多系统都十分友好的数据结构，而使用空格和特殊字符（即a~z、0~9除外的其他字符）忽略了*可移植性*。我们在第1章中将这一专业术语定义为"以一种双方系统都兼容的方式在平台间传递信息"。如果我们这么做的话，会直接降低JSON数据的可移植性；因此我们说，为了获得*最大可移植性*，应尽可能避免使用空格或特殊字符。

JSON中的名称－值对的名称如果被系统作为对象装入内存的话，将会成为"属性"。在部分系统中，属性名可以包含下划线（_）或数字，但是大多数情况下最好是使用英文字母A~Z或a~z。所以，当我想在我的名称中使用多个单词时，我一般会这样写：

```
"lindsaysAnimal": "cat"
```

或：

```
"myAnimal": "cat"
```

示例中的"cat"值是被双引号包裹的。不过，不同于名称，值并不总是需

要被双引号包裹。当值是字符串时，必须使用双引号。而在 JSON 中，还有数字、布尔值、数组、对象、null 等其他数据类型，这些都不应被双引号包裹。这些格式将在第 3 章讲解。

JSON 的全称是 JavaScript 对象表示法。所以我们剩下的任务就是了解构建一个对象的语法。我们将花括号加在名称－值对的两边来使之成为一个对象。像下面这样，一个在前，一个在后：

```
{ "animal" : "cat" }
```

当你在格式化 JSON 时，不妨想象一下骑士的册封仪式，封主会用剑在受封者的左右肩各轻点一下，使他自此成为一名真正的骑士。而现在你就是这个封主，你需要在两边点上花括号来使你的 JSON 成为一个对象。“JSON 先生，我在此封你为对象。”如果不在左右肩上用剑轻点，那么仪式便不是完整的。

在 JSON 中，多个名称－值对使用逗号来分隔。现在，让我们来扩充“animal/cat”示例，加入一种颜色：

```
{ "animal" : "cat", "color" : "orange" }
```

我们也可以从读它的机器的角度去理解 JSON 的语法。和人不同，机器是严格遵守规则和指令的。当你在一个字符串类型的值外面使用下面这些字符时，实际上提供的是告诉机器如何读取数据的指令：

- {（左花括号）指“开始读取对象”
- }（右花括号）指“结束读取对象”
- [（左方括号）指“开始读取数组”
-]（右方括号）指“结束读取数组”
- :（冒号）指“在名称－值对中分隔名称和值”
- ,（逗号）指“分隔对象中的名称－值对”或者“分隔数组中的值”；也可以认为是“一个新部分的开始”

如果你没有用右花括号来“结束读取对象”，那么你的对象将不会被解析成对象。如果你在名称－值对列表的结尾处加上一个逗号，你给机器的指令是“一个新部分的开始”，但是后面什么都没有。因此，时刻使用正确的语

法十分重要。

小故事：JSON 的双引号

有一天，我无意间瞥见了一个学生的电脑屏幕。屏幕上显示着他即将拿去验证的 JSON（示例 2-2）。

示例 2-2：不能通过验证的“JSON”

```
{
    title : "This is my title.",
    body : "This is the body."
}
```

验证后，验证器抛出了一个解析错误。他感到很困惑，说：“你看嘛，我写得没有错啊！”

我告诉他 "title" 和 "body" 的两边没有加上双引号。他说：“但是我见过在名称两边加引号和不加引号的两种 JSON 格式。”“哈，”我说，“没有在名称两边加上双引号时，它并不是 JSON，而是一个 JavaScript 对象。”

有这种困惑是可以理解的。JSON 是基于 JavaScript 对象字面量的，所以它们看起来很像。但是 JavaScript 对象字面量不需要给名称 – 值对中的名称两边加上双引号。而在 JSON 中，却是必须要加的。

另外一个让人困惑的点是使用单引号代替双引号。在 JavaScript 中，一个对象允许使用单引号代替双引号（示例 2-3）。

示例 2-3：这不是合法的 JSON

```
{
    'title': 'This is my title.',
    'body': 'This is the body.'
}
```

在 JSON 中，我们仅使用双引号，而且对于名称 – 值对中的名称来说，它们是必需的（示例 2-4）。

示例 2-4：合法的 JSON

```
{
    "title": "This is my title.",
    "body": "This is the body."
}
```

2.4 语法验证

和机器不同，对我们这些敲键盘的人来说，只要少敲个字就能酿成错误。我们没有创造比想象中更多的错误，真的是很神奇。所以当你在工作中使用 JSON 时，很重要的一点就是验证。

你使用的集成开发环境（integrated development environment，IDE）也许会内置 JSON 的验证。如果内部没有集成，但是你的 IDE 支持插件的话，你应该也能找到相关的验证工具。如果你不使用 IDE，甚至对我刚刚说的都一无所知，这也没有关系。

有许多在线工具可以帮助你格式化和验证 JSON。在你常用的搜索引擎中搜索"JSON 验证"就能得到许多结果。下面是一些值得一提的工具。

- JSON Formatter & Validator（https://jsonformatter.curiousconcept.com/）
  这是一个带有配置选项、能够高亮错误且 UI 很棒的格式化工具。经过处理的 JSON 会显示在两个窗口，一个用于展示 JSON 的树 / 节点结构，类似于可视化工具，另一个用于复制 / 粘贴格式化后的代码。

- JSON Editor Online（http://www.jsoneditoronline.org/）
  这是一个集验证、格式化和可视化工具于一身的 JSON 工具。错误提示会显示在出错的那一行。除了验证以外，还会显示解析错误的详情。右边的可视化工具使用树 / 节点的形式来展示 JSON。

- JSONLint（http://jsonlint.com/）
  这是一个毫不花哨的 JSON 验证工具。简单地复制、粘贴、验证即可。也可以友好地格式化你的 JSON。

以上这些都是*语法验证*工具。我们会在第 4 章讨论另一种类型的验证——一致性验证。语法验证关注的是 JSON 的格式，而一致性验证关注的是其独特的数据结构。比如我们对示例 2-5 进行验证，语法验证会检测我们的 JSON 语法是否正确（是否被花括号包裹，名称 – 值对是否以逗号分隔），而一致性验证会检测我们的数据中是否包含 name、breed 和 age 等信息。它还会检测 age 的值是不是数字，name 的值是不是字符串，等等。

示例 2-5：验证示例

```
{
    "name": "Fluffy",
    "breed": "Siamese",
    "age": 2
}
```

2.5 JSON文件

你可能会觉得在今后使用 JSON 时，仅能在代码中创建它并传输到一个仅可通过开发者工具来查看的不可见的世界。然而，JSON 这种数据交换格式是可以作为独立的文件存在于文件系统中的。它的文件扩展名非常好记：.json。

因此，我可以将“animal/cat”保存到计算机中的一个 JSON 文件中，比如 C:/animals.json。

2.6 JSON的媒体类型

当你在传递数据时，需要提前告知接收方数据是什么类型，这就会涉及媒体类型。媒体类型也有一些你可能听过的其他称呼，如“互联网媒体类型”“内容类型”或“MIME 类型”。它使用“类型 / 子类型”这种格式来表示，比如你可能见过的 `text/html`。

JSON 的 MIME 类型是 `application/json`。

互联网数字分配机构（Internet Assigned Numbers Authority，IANA）维护着一个包含全部媒体类型的列表（http://www.iana.org/assignments/media-types/media-types.xhtml）。

2.7 专业术语和概念

本章涵盖了以下专业术语。

- 字面量
  指字面意思与其想要表达的意思是完全一致的值。

- *变量*

  通过形如 *x* 的标识符来表示的、可以修改的一类值。

- *最大可移植性（数据交换中）*

  通过保证数据本身对于平台和系统的兼容性来提供超越数据格式本身的可移植性。

- *名称－值对*

  指拥有名称和对应的值的属性或特征（也叫键－值对）。

- *语法验证*

  关注 JSON 格式的验证。

- *一致性验证*

  关注独特数据结构的验证。

我们还讨论了以下重要概念。

- JSON 基于 JavaScript 对象字面量中表示属性的语法，但是并不包含与 JavaScript 对象字面量的函数相关的部分。
- 在 JSON 的名称－值对中，名称始终被双引号包裹。
- 在 JSON 的名称－值对中，值可以是字符串、数字、布尔值、null、对象或数组。
- JSON 中的名称－值对列表被花括号包裹。
- 在 JSON 中，多个名称－值对使用逗号分隔。
- JSON 文件使用 .json 扩展名。
- JSON 的媒体类型是 `application/json`。

第 3 章

JSON的数据类型

如果你已经学过一两门编程语言的话，对数据类型肯定有所了解。当然，没学过也没有关系，让我们来简单介绍一下。

3.1 数据类型简介

想象一下，如果把一把锤子交给一个对工具一无所知的孩子，也不告诉他用途的话，会发生什么。这很可能会危害到他人的生命和财产安全。不过，如果这个孩子循规蹈矩、身体协调，我们可以就告诉他应该如何使用锤子。这样他就会用它来钉钉子和抠钉子，而不会拿着它四处乱跑破坏东西（毕竟这是个乖孩子）。而且，你跟他说“把锤子递给我”，他肯定不会给你螺丝刀。提前了解并学会使用一样事物是很有用的，无论是在计算世界中，还是在现实世界中。

在计算机中，我们需要知道正在处理什么类型的数据，因为不同类型的数据有着不同的操作途径。可以让两个数相乘，但是不能让一个单词和一个数相乘。如果我有一个单词表，可以按字母顺序给它们排序。但是数字 5 可没有字母顺序。所以在编程中，当一个方法（或函数）说“请给我传递一个数字”时，如果我们知道什么是数字的话，就不会错把单词“ketchup”传给它。

在计算机科学中，有一种数据类型被称为原始数据类型。“原始”这个词可能会让我们联想到石器时代的原始人围坐在篝火旁低语交流、磨木棒一类的场景。但是，这里所指的数据类型可不是像原始人那样粗陋的数据，确切地说，它们指的是数据中第一位的、最基本的一种类型。就像现代人进化自原始人一样，许多现在常用的数据类型都是基于以下这些原始数据类型的：

- 数字（如 5 或 5.09）
  - 整型
  - 浮点数
  - 定点数
- 字符和字符串（如“a”“A”或“apple”）
- 布尔类型（即真或假）

在不同的编程语言中，这类“一成不变”的数据类型常被叫作原始数据类型或内置类型。这意味着它们的定义和操作都是不能修改的。比如，编程语言不会允许你重新定义两数相加的含义。另外，由于语言的不同，原始数据类型也有所不同，除了上面列举的以外，还可能有字节类型或引用类型（或指针、句柄）等。

除了原始数据类型，大多数编程语言中还有许多其他的类型。它们常常被称为复合数据类型，因为它们是由原始数据类型融合而成的。复合数据类型就像一个沙堡，是有自身结构的。如果我们把沙堡拆开，会发现它是由沙子、木棒和水构成的。如果我们拆开复合数据类型的数据结构，同样会发现它是由原始数据类型所构成的。

枚举数据类型是编程语言中常见的复合数据类型之一。之前我提到过按字符表顺序为单词表排序。单词表在不同的编程语言中有着不同的数据类型（如列表或数组）。如果我们把这一数据结构拆开，会发现它可能是由字符或字符串这种原始数据类型构成的。枚举数据类型，就是一个可以枚举的数据结构。我可以把这一结构中的东西一个一个拿出来，并且可以算出它们的总数。见示例 3-1。

示例 3-1：“让我们来枚举你的优良品德”可以用数组字面量的形式在编程语言中表示

```
[
    "witty",
    "charming",
    "brave",
    "bold"
]
```

你不需要理解示例中的数组字面量这种数据结构，就能发现我们可以挨个列举这些“优良品德”，并且它们的总数是四。

另一种复合数据类型是对象数据类型。我们在第 2 章已经讨论过它，因为 JSON 是基于 JavaScript 中的对象字面量表示法的。此外，我还用 JSON 描述了我的鞋子（见示例 3-2）。

示例 3-2：使用 JSON 描述的我的鞋子

```
{
    "brand": "Crocs",
    "color": "pink",
    "size": 9,
    "hasLaces": false
}
```

通过对象字面量，我们能够发现对象数据类型是由名称－值对所构成的。如果进一步对这一数据类型进行解构，我们会发现它所使用的原始数据类型：字符串、数字和布尔值。名称－值对中的这些名称（`"brand"`、`"color"`、`"size"` 和 `"hasLaces"`）都是由字符串构成的。`"Crocs"` 和 `"pink"` 这两个值也都是字符串。`"9"` 这个值是数字。`"false"` 这个值是布尔类型。

3.2 JSON中的数据类型

虽说对于复合数据类型，乃至于一小部分原始类型来说，它们的编程语言存在许多差异，但我最开始提到的原始类型，大多数语言中都是涵盖的：

- 数字（如 5 或 5.09）
  - 整型
  - 浮点数

  - 定点数
- 字符和字符串（如“a”“A”或“apple”）
- 布尔类型（即真或假）

对象数据类型是在大多数编程语言中都很常见且流行的数据类型，如 Java 或 C#，不过不是全部。由于 JSON 基于对象字面量表示法以及对象数据类型，你可能觉得让它作为数据交换格式有些不妥。毕竟数据交换格式是以让不同的两个系统间能够进行交流为目标的，这一格式所表达的必须是共有的部分。记住，复合数据类型对象的数据结构可以被解构为原始数据类型。即便是对那些不支持对象数据类型的语言来说，一旦这一数据结构被解构为那些原生的类型，就很好处理了。

JSON 中的数据类型包括：

- 对象
- 字符串
- 数字
- 布尔值
- null
- 数组

3.3 JSON中的对象数据类型

JSON 中的对象类型非常简单。追根溯源，JSON 本身就是对象，也就是一个被花括号包裹的名称 – 值对的列表。如果你希望在作为对象的 JSON 中创建一个名称 – 值对，那就需要用到嵌套。示例 3-3 向我们展示了如何用嵌套对象来描述一个人。

示例 3-3：嵌套对象

```
{
    "person": {
        "name": "Lindsay Bassett",
        "heightInInches": 66,
        "head": {
            "hair": {
                "color": "light blond",
                "length": "short",
```

```
                "style": "A-line"
            },
            "eyes": "green"
        }
    }
}
```

本例中，最高一级的名称－值对是 person，其值是一个对象。这一对象中有三个名称－值对："name"、"heightInInches" 和 "head"。"name" 这个名称－值对的值是 "Lindsay Bassett"。"heightInInches" 的值是一个数字。"head" 的值又是一个对象。其中 "hair" 的值同样是一个对象。这一对象中包含了三个名称－值对："color"、"length" 以及 "style"。"head" 的对象中还有一个名称为 "eyes" 的名称－值对，其值为 "green"。

3.4 JSON中的字符串类型

前面我们曾通过"animal/cat"这个示例简单讨论了 JSON 中的字符串类型：

```
{ "animal" : "cat" }
```

这里的 "cat" 就是一个字符串类型的值。实际应用中，我们不可能遇到这么简单的字符串类型的值，除非是宠物店的数据。甚至连宠物店也不可能在它的数据中仅仅使用"cat"这么简单的单词。比如，商店想要发布它的促销详情：

> 今天，如果你在 Bob's Best Pets 购买一只小狗，可免费获得一袋价值 8 盎司的 Bill 狗粮。你只需在结账时说"Bob's 是最棒的！"即可。

JSON 中的字符串可以由任何 Unicode 字符构成，因此上面的促销详情中的所有字符都是可以使用的。字符串的两边必须被双引号包裹。

在第 2 章的"小故事：JSON 的双引号"中，我提到过使用单引号来包裹字符串值是不合法的（示例 3-4）。

示例 3-4：这不是一个合法的 JSON

```
{
    'title': 'This is my title.',
```

```
        'body': 'This is the body.'
    }
```

这一点确实容易让人犯迷糊，尤其是在你之前接触过 JavaScript 对象字面量的情况下。在 JavaScript 中，使用单引号或双引号没有任何区别。然而请记住，JSON 不是 JavaScript 对象字面量，它只是基于 JavaScript 对象字面量。而在 JSON 中，仅允许使用双引号来包裹字符串。

我们还提到过解析器是如何读 JSON 的。对解析器来说，当一个值以双引号（"）开始时，它希望接下来的字符串文本以另一个双引号结尾。这意味者如果这段字符串本身包含双引号可能会出错。

举个例子，假如我们就是那个正在进行促销的宠物店店主，我们希望顾客在结账时说出“Bob’s 是最棒的！”来获得一袋免费狗粮。如果我们使用下面示例 3-5 中的代码，会出现一个问题，因为这时不能仅用一对双引号来包裹促销数据。

示例 3-5：这段代码不能正常使用

```
{
    "promo": "Say "Bob's the best!" at checkout for free 8oz bag of
    kibble."
}
```

由于在值里面有双引号，解析器在读取第一个双引号之后，会把“Bob”前面的双引号当成字符串结尾的双引号。然后解析器发现后面还有许多不属于任何一个名称 – 值对的文字，就会报错。为了处理这个问题，我们需要在字符串中的双引号前面加上一个反斜线字符来对其转义，详见示例 3-6。

示例 3-6：使用反斜线对字符串中的双引号进行转义来解决这一问题

```
{
    "promo": "Say \"Bob's the best!\" at checkout for free 8oz bag
    of kibble."
}
```

反斜线会告诉解析器这个双引号并不意味着字符串的结束。一旦解析器将字符串装入内存，每一个双引号前的反斜线都会被移除，文本也会按预期输出。

对于 JSON 的字符串来说，双引号也不是唯一一个需要转义的字符。因为反斜线用于转义字符，所以我们还需要转义反斜线。例如，示例 3-7 中的

JSON 内容为我的 Program Files 文件夹的路径，会报错。为了解决这个问题，我们需要给这个反斜线加上另外一个反斜线来进行转义，详见示例 3-8。

示例 3-7：在这段代码中使用反斜线会报错

```
{
    "location": "C:\Program Files"
}
```

示例 3-8：反斜线需要另一个反斜线来转义

```
{
    "location": "C:\\Program Files"
}
```

除了双引号和反斜线，还需要转义以下字符：

- `\/`（正斜线）
- `\b`（退格符）
- `\f`（换页符）
- `\t`（制表符）
- `\n`（换行符）
- `\r`（回车符）
- `\u` 后面跟十六进制字符（如笑脸表情 `\u263A`）

示例 3-9 中的 JSON 会由于没有对制表符和换行符进行转义而报错，示例 3-10 展示了如何解决这一问题。[1]

示例 3-9：在 JSON 中使用制表符和换行符会报错

```
{
    "story": "\t Once upon a time, in a far away land \n there lived
    a princess."
}
```

示例 3-10：对制表符和换行符转义后的 JSON

```
{
    "story": "\\t Once upon a time, in a far away land \\n there
    lived a princess."
}
```

注 1：在使用网上的 JSON 验证器后发现上述代码并不会报错，示例 9 和示例 10 表达的实际上是不同的意思。——译者注

3.5 JSON中的数字类型

数字是一种常见的用于传递数据的信息片段。库存数目、金额、经度 / 纬度以及地球的质量等均可以用数字来表示，详见示例 3-11。

示例 3-11：JSON 中的数字表示

```
{
    "widgetInventory": 289,
    "sadSavingsAccount": 22.59,
    "seattleLatitude": 47.606209,
    "seattleLongitude": -122.332071,
    "earthsMass": 5.97219e+24
}
```

JSON 中的数字可以是整数、小数、负数或者指数。

商品库存数是 289。库存是一个典型的整数。我肯定不会卖出半个商品，所以库存数不可能有小数点。

我可怜的账户里只有 22.59 美元。尽管在一些编程语言中有专门针对金额的数据类型，但在 JSON 中通常使用小数来表示金额，并省略单位。

如果你注意一下西雅图市的经度和纬度，会发现它们都是小数，且纬度是负数。负数只需在数字前面加上标准的负号字符即可。

此外，我用一个超大的数来表示地球的质量，单位为千克，并使用了指数表示法。指数表示法常常出现在科学数据中，而且也是 JSON 支持的数字。

3.6 JSON中的布尔类型

在口语中，对问题最简单的回答莫过于肯定或否定。如果你问朋友“你的煎蛋要不要配上吐司面包”，他就会回答“要”或“不要”。

在计算机编程中，布尔类型是很简单的。它不是真就是假。如果你问你的电脑“你的煎蛋要不要配上吐司面包”，它就会回答“真”（true）或“假”（false）。

在一些编程语言中，`true` 的字面值可能用 1 来表示，`false` 用 0 来表示。

有时候字面值也可能是大写或小写的单词，比如 `true` 或 `TRUE`，`false` 或 `FALSE`。在 JSON 中，该字面值仅使用小写形式：`true` 或 `false`，任何其他形式的写法都会报错。示例 3-12 中，我使用布尔值来描述对早餐和午餐喜好的数据。

示例 3-12：喜好

```
{
    "toastWithBreakfast": false,
    "breadWithLunch": true
}
```

3.7 JSON中的null类型

对于一无所有的东西，你可能觉得用 0 来描述比较合适。比如，我有 0 个手表。但事实是，0 是一个数字。这意味着本质上是在计数。

如果用一种 JSON 的标准格式来描述一个人手腕的情况，会是怎样的呢？详见示例 3-13 及示例 3-14。

示例 3-13：隔壁 Bob 可能是这样

```
{
    "freckleCount": 0,
    "hairy": true,
    "watchColor": "blue"
}
```

示例 3-14：我的可能是这样

```
{
    "freckleCount": 1,
    "hairy": false,
    "watchColor": null
}
```

因为我不戴手表，所以自然不会有手表颜色。而在编程中，`null` 就用来表示 0、一无所有、不存在等意思，而不用数字来表示。由于手表颜色的值也是不能被定义的，所以使用 `null` 来描述。

不要把 `null` 和 `undefined` 混淆，尤其是在使用 JavaScript 时。`undefined` 不是 JSON 中的数据类型，而在 JavaScript 中，`undefined` 是在尝试获取一些

不存在的对象或变量时返回的结果。在 JavaScript 中，`undefined` 与那些声明的名称和值都不存在的对象或变量有关，而 `null` 则仅与对象或变量的值有关。`null` 是一个表示“没有值”的值。在 JSON 中，`null` 必须使用小写形式。

3.8 JSON中的数组类型

现在探讨一下数组数据类型。如果你对数组不熟悉也没关系，我们先来简单介绍一下。

想象一个装着一打鸡蛋的容器。容器里有 12 个位置用来放鸡蛋。我刚买下这些鸡蛋时，数量是 12 个。这就相当于有一个大小为 12 的数组，包含 12 个鸡蛋，见示例 3-15。

示例 3-15：这是一个字符串数组（为了简便起见，使用字符串 `"egg"` 来表示每一个位置上的鸡蛋）

```
{
    "eggCarton": [
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg"
    ]
}
```

注意，这是一个名称 – 值对。其名称是 `"eggCarton"`，值是一个数组。数组始终应被方括号（`[]`）包裹。在数组中，可以看到一个列表，列表项之间用逗号隔开。这有点像处理名称 – 值对的方式，不过一个关键的区别是这个列表里只有值。这些值可以是任何合法的 JSON 数据类型（字符串、数字、对象、布尔值、数组以及 `null`）。

现在，假如我拿出两个鸡蛋做成煎蛋来搭配早餐的吐司面包。鸡蛋盒里还是有 12 个位置，不过有两个鸡蛋被拿走了。详见示例 3-16。

示例 3-16：从蛋箱中拿出两个鸡蛋做早餐

```
{
    "eggCarton": [
        "egg",
        null,
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        null,
        "egg"
    ]
}
```

你可以看到，我拿走了两个位置上的鸡蛋，于是这两个位置就是空的了。我们把它们用 `null` 来表示。

一个数组为每一个“位置”都赋予了一个索引。我们从 0 开始，那么第一个位置的序号便是 0，第二个位置的序号是 1，以此类推。最后一个位置的序号是 11。所以，那两个被拿走鸡蛋的位置的序号是 1 和 10。这是一个合法的数组。

如果在 10 的位置上放上数字 5，那么在大多数编程语言中，这样的数组都是不合法的。详见示例 3-17。

示例 3-17：这在大多数语言中都是不合法的

```
{
    "eggCarton": [
        "egg",
        null,
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
        "egg",
```

```
            "egg",
            "egg",
            5,
            "egg"
        ]
    }
```

虽然在“大多数语言”中不合法，但在 JSON 中，这种混合使用数据类型的情况是合法的。接下来我会告诉你为什么合法，以及为什么应该避免这样做。

在 JavaScript 中，你定义了一个变量。例如，示例 3-18 中有一个名为 `something` 的变量，我们将其赋值为数字 5。

示例 3-18：在 JavaScript 中定义变量

```
var something = 5;
```

紧接着把这一变量的值改为一个字符串（示例 3-19）。

示例 3-19：将这一变量的值改为字符串

```
something = "bob";
```

接下来再把它的值改成一个对象（示例 3-20）。

示例 3-20：将这一变量的值改成一个对象

```
something = { person: "bob" };
```

使用 `var` 声明的变量 `something` 的值可以是数字、字符串、数组、`null` 以及对象中的任意一种类型。不过大多数语言都不允许变量的类型随意改变。正常情况下，在声明变量时，也需要说明它们是整型、字符串还是对象。所以在声明 `something` 这个变量时，得用 `int something = 5`、`string something = "bob"` 或是 `Person something = new Person("Bob")`。所以在大多数语言中，当声明一个数组时，也要声明所有容器中所保存的数据都应是什么类型，且之后不能随意修改。

JSON 是一种数据交换格式。如果将 JSON 数据传递给一个不使用 JavaScript 的系统，那么在解析时很可能会出错。

这就好比去参加一个奇石藏品展卖会。你出售一套含 50 块奇石的藏品，一

个哥们过来从你这里买了这套藏品，但是到家一看，发现这一套藏品并不是 50 块奇石，而是 20 块奇石、20 根木棒和 10 块口香糖（有些还被嚼过）。

下面仔细看一些各种数据类型的数组的例子。在 JSON 中，数组里可以包含任何支持的数据类型。所以可以有字符串构成的数组、数字构成的数组、布尔值构成的数组、对象构成的数组，甚至是数组构成的数组。数组构成的数组也被称为多维数组。下面是一些例子。

假如有一个学生上课的签到名单。这可以用字符串构成的数组表示（示例 3-21）。

示例 3-21：使用字符串构成的数组来表示学生签到表

```
{
    "students": [
        "Jane Thomas",
        "Bob Roberts",
        "Robert Bobert",
        "Thomas Janerson"
    ]
}
```

在学生们考完试以后，可以用数字构成的数组来表示他们的分数（示例 3-22）。

示例 3-22：使用数字构成的数组来表示考试分数

```
{
    "scores": [
        93.5,
        66.7,
        87.6,
        92
    ]
}
```

如果需要给只包含判断题的考试做一份答案，可以使用布尔值构成的数组（示例 3-23）。

示例 3-23：使用布尔值构成的数组来表示判断题的答案

```
{
    "answers": [
        true,
```

```
            false,
            false,
            true,
            false,
            true,
            true
        ]
    }
```

一个由对象构成的数组可以表示整个考试，既包括问题也包括答案（示例3-24）。

示例 3-24：使用由对象构成的数组来表示一场考试的问题和答案

```
{
    "test": [
        {
            "question": "The sky is blue.",
            "answer": true
        },
        {
            "question": "The earth is flat.",
            "answer": false
        },
        {
            "question": "A cat is a dog.",
            "answer": false
        }
    ]
}
```

为了表示三场不同考试的答案，可以使用由数组构成的数组，也就是多维数组来解决（示例 3-25）。

示例 3-25：使用由数组构成的数组来表示三场不同考试的答案

```
{
    "tests": [
        [
            true,
            false,
            false,
            false
        ],
        [
            true,
            true,
```

```
                true,
                true,
                false
            ],
            [
                true,
                false,
                true
            ]
        ]
    }
```

3.9 专业术语和概念

本章涵盖了以下专业术语。

- JSON 中的字符串类型
  一个字符串值，如 "my string"，使用双引号包裹。
- JSON 中的布尔类型
  true 或 false。
- JSON 中的数字类型
  一个数字值，如 42，可以是正整数、负整数、小数或指数。
- JSON 中的 null 类型
  一个表示空值的 null 值。
- JSON 中的数组类型
  数组是值的集合或列表，每个值都可以是字符串、数字、布尔值、对象或数组中的任何一种，数组必须被方括号（[]）包裹，且值与值之间用逗号隔开。
- JSON 中的对象类型
  对象类型是使用逗号分隔的名称 – 值对构成的集合，并使用花括号（{}）包裹。

我们还讨论了以下重要概念。

- JSON 中布尔类型的值只有 `true` 和 `false`，且所有字母必须小写（即 `true`，不能是 `True` 或 `TRUE`）。
- JSON 中的 `null` 值的所有字母必须小写（即 `null`，不能是 `NULL` 或 `Null`）。
- 对象和数组很关键的一个区别就是，对象是名称－值对构成的列表或集合，数组是值构成的列表或集合。
- 对象和数组另一个关键的区别是，数组中所有的值应具有相同的数据类型。

第 4 章

JSON Schema

第 3 章讲解了 JSON 的数据类型，并对其重要性和应用价值进行了探讨。是否提前了解事物的本质和用途（想想那个拿着锤子的孩子）会对事情的进展产生很大的影响。

在大多数情况下，数据交换格式中的数据通过互联网或其他网络传输到接收方。而接收方会对数据文件的格式，包括其结构和数据类型有一个预期。所以，接收方通常会提供一个文档来解释预期的格式并提供示例。

然而无论提供的文档多么详尽，数据都有可能会出错。要说明一点，我们在这里要探讨的不是语法错误，而是一些诸如“我寄了一个苹果，但你想要的是橘子”这类出于误解的错误。本书将这类验证称为一致性验证（conformity validation），以此与语法验证区分开来。

在这里，这一验证过程通常分为以下几步。

(1) 你创建好了数据并且信心十足。
(2) 你通过互联网向接收方发送数据。今天网速很慢，你发送的数据文件又很大，所以这花了几分钟的时间。
(3) 由于数据不符合接收方的要求，你只收到了错误响应。虽说信心备受打击，但幸运的是，错误响应告诉了你一些有价值的信息，比如错在哪里。
(4) 你重新查阅文档，找到并修复你认为出错的地方，然后重回步骤 (1)。

早在 JSON 出现以前，这样的情况就已经存在于数据交换之中。幸运的是，技术从业者都是擅长解决问题的人，Schema（意为模式）这一概念也随之诞生。

4.1　验证的魔力

在现实生活中，对于重要事项，常常需要双方签订合同。当签订一份要求完成某个项目的合同时，合同会对具体的细节进行概述。比如，我同意在 8 月 31 日前交付宇宙飞船，最终的产品将是一个功能健全的宇宙飞船，包含完整的生命支持系统、光照系统以及三个引擎。

想象我们现在生活在一个魔法世界。当公司将项目合同交给我时，在上面施了魔法。任何时候，只要我用魔杖轻点合同书，它就会告诉我是否完成了任务。这就避免了一些尴尬的情况，比如我在会上告诉大家“完工了”，结果有人说“你答应放在飞船上的第三个引擎去哪了？”。我随时都可以验证是否真的完成了任务，完成后就可以充满信心地去开会。

数据交换 Schema 和这个假想世界中的魔法很相似。在发送数据前，我们随时可以验证数据是否与 Schema 一致，以便知道会不会被接收方接受。当在交换数据的过程中使用 Schema 时，所经历的步骤和不使用 Schema 时有很大的差别，如下所示。

(1) 通过验证数据与 Schema 的一致性，你轻松地定位和修复了错误。你从每个错误中都能得到许多有用的信息。
(2) 创建好了数据并且信心十足。
(3) 你从网上将数据发送过去，并收到了成功响应。任务完成。

此外，JSON Schema 也可以被接收方用于传输的另一端。JSON Schema 可以位于要接收的数据的第一行，以保证数据符合要求。它可以在数据被处理前回答下面的所有问题。

- 值的数据类型是否正确?
  可以具体规定一个值是数字、字符串等类型。

- 是否包含所需要的数据？
  可以具体规定哪些数据是需要的，哪些是不需要的。
- 值的形式是不是我需要的？
  可以指定范围、最小值和最大值。

4.2 JSON Schema简介

尽管 JSON 已经相当成熟，但 JSON Schema 仍在开发之中。截至 2015 年 4 月，JSON Schema 最新版是草拟版本 4。当然这并不意味着你现在不可以使用它，这仅仅说明了它仍在成长，且将来会做得更好。

JSON Schema 使用 JSON 来书写，所以仅需几步就能掌握它。首先，需要在 JSON 第一个名称 – 值对中，声明其为一个 schema 文件（示例 4-1）。

示例 4-1：声明的名称必须为 "$schema"，值必须为所用草拟版本的链接

```
{
    "$schema": "http://json-schema.org/draft-04/schema#"
}
```

第二个名称 – 值对应该是 JSON Schema 文件的标题（示例 4-2）。

示例 4-2：表示一只猫的 JSON Schema 文件格式

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Cat"
}
```

在 JSON Schema 文件的第三个名称值对中，要定义需要在 JSON 中包含的属性。"properties" 的值实质上是我们想要的 JSON 的名称 – 值对的骨架。我们没有使用字面量，而是使用了一个对象来定义数据类型，并有选择地进行描述（示例 4-3）。

示例 4-3：定义猫的属性

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Cat",
  "properties": {
    "name": {
```

```
      "type": "string"
    },
    "age": {
      "type": "number",
      "description": "Your cat's age in years."
    },
    "declawed": {
      "type": "boolean"
    }
  }
}
```

接下来就能验证 JSON 是否符合 JSON Schema（示例 4-4）。

示例 4-4：这个 JSON 符合针对猫定义的 JSON Schema

```
{
  "name": "Fluffy",
  "age": 2,
  "declawed": false
}
```

前面提到过，JSON Schema 可以帮助回答下列问题。

- *值的数据类型是否正确？*
  可以具体规定一个值是数字、字符串等类型。

- *是否包含所需要的数据？*
  可以具体规定哪些数据是需要的，哪些是不需要的。

- *值的形式是不是我需要的？*
  可以指定范围、最小值和最大值。

在猫的示例中，我们解决了第一个问题。可以验证表示“Fluffy”这只猫的 JSON 中的 `name`、`age` 和 `declawed` 是否都有正确类型的值。接下来处理第二个问题：是否包含所需要的数据？

对于数据，总有一些属性（或字段）必须包含值，也有一些不用包含。例如，在一家购物网站上注册一个新账户，需要填写详细的收货地址，其中必填项是姓名、省、市、街道和邮政编码。还可以选填公司名、小区名、门牌号，等等。如果遗漏了必填字段，那么将无法进行后续步骤。

为了在 JSON Schema 中实现这一逻辑，需要在 "$schema"、"title" 和 "properties" 后面加上第四个名称 – 值对，它的名称是 "required"，值为一个数组。数组中包含必填的字段。

在示例 4-5 中，先加入一个新的 "description" 字段。接下来加上第四个名称 – 值对 "required"，先让它的值为一个空数组。由于 "name"、"age" 和 "declawed" 是必填字段，所以把它们加入数组。"description" 不是必填字段，就不应加入数组中。

示例 4-5：定义必填字段

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Cat",
  "properties": {
    "name": {
      "type": "string"
    },
    "age": {
      "type": "number",
      "description": "Your cat's age in years."
    },
    "declawed": {
      "type": "boolean"
    },
    "description": {
      "type": "string"
    }
  },
  "required": [
    "name",
    "age",
    "declawed"
  ]
}
```

在将 "required" 字段加入 JSON Schema 后，合法的 JSON 如示例 4-6 所示。这段 JSON 中包含了在 Schema 中定义的 "name"、"age" 和 "declawed" 三个字段，也包含了可选的 "description" 字段。

示例 4-6：合法的 JSON

```
{
  "name": "Fluffy",
  "age": 2,
```

```
    "declawed": false,
    "description" : "Fluffy loves to sleep all day."
  }
```

由于 "description" 不属于必填字段，所以也可以不填它。示例 4-7 也同样符合我们定义的关于猫的 JSON Schema，它包含了必填字段 "name"、"age" 和 "declawed"。

示例 4-7：不包含 "description" 字段的合法 JSON

```
{
  "name": "Fluffy",
  "age": 2,
  "declawed": false
}
```

非常重要的一点是，如果你的 JSON Schema 中不包含 "required" 名称 - 值对，那么将不会有必填项。一个没有任何名称 – 值对的空 JSON 对象也被认为是合法的。如果没有 "required" 数组，那么示例 4-8 也会被认为是符合关于猫的 JSON Schema。

示例 4-8：合法的 JSON

```
{}
```

我们可以利用 JSON Schema 回答的第三个，也是最后一个问题是：值的形式是不是我需要的？虽然前面解决了数据类型的问题，但我们常常需要对该类型的格式提出更具体的要求。比如，我需要一个用户名，但用户名的长度不能超过 20 个字符。另外，还有可能要求数字的范围是 10~100。这些具体的要求也都可以在 JSON Schema 中体现。

在猫的 JSON 示例中，虽然提出了一些诸如名字应为字符串、年龄应为数字之类的要求，但我们肯定不希望有人给小猫起过长或过短的名字，也不希望把年龄填成负数。所以在 JSON Schema 中可以定义字符串长度的最小值和最大值，以及数字的最小值。

在示例 4-9 中，我们增加了新的验证项目来保证猫的名字长度不少于 3 个字符，也不多于 20 个字符。此外，还保证了小猫的年龄不为负数。

示例 4-9：验证猫的 JSON

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Cat",
  "properties": {
    "name": {
      "type": "string",
      "minLength": 3,
      "maxLength" : 20
    },
    "age": {
      "type": "number",
      "description": "Your cat's age in years.",
      "minimum" : 0
    },
    "declawed": {
      "type": "boolean"
    },
    "description": {
      "type": "string"
    }
  },
  "required": [
    "name",
    "age",
    "declawed"
  ]
}
```

示例 4-10 中的 JSON 由于名称值长度超过了 `"maxLength"` 且年龄小于 `"minimum"`，因而不会通过验证。

示例 4-10：不能通过验证的 JSON

```
{
  "name": "Fluffy the greatest cat in the whole wide world",
  "age": -2,
  "declawed": false,
  "description" : "Fluffy loves to sleep all day."
}
```

示例 4-11 向我们展示了一个符合猫的 JSON Schema 要求并符合对值的要求的合法 JSON。

示例 4-11：合法的 JSON

```
{
  "name": "Fluffy",
  "age": 2,
  "declawed": false,
  "description" : "Fluffy loves to sleep all day."
}
```

如果再回过头来比较 Schema 和合同，会发现合同中所规定的细节可以非常具体。本章提供的例子只是简介，是冰山一角。JSON Schema 还支持正则表达式（一种字符形式，比如电子邮件地址的格式）以及枚举类型（一个包含所有可能值的列表）。如果你希望深入掌握 JSON Schema，可以访问下面的链接，了解相关规范：

- JSON Schema 主页（http://json-schema.org/）
- JSON Schema 验证规范（http://json-schema.org/latest/json-schema-validation.html）

现在有许多针对具体编程语言和框架的 JSON Schema 库和项目，并且还在增加。如果你想在你的项目中使用 JSON Schema，那么搜索"JSON Schema 验证 [在这里输入具体的编程语言名称]"会帮你找到你所需要的东西。此外，还有一些在线验证工具，它们都与编程语言无关，也是进行 JSON Schema 验证的好帮手：

- JSON Schema Lint（http://jsonschemalint.com/draft4/）
- JSON Schema Validator（http://www.jsonschemavalidator.net/）

JSON Schema Lint 这个网站包含两个文本区域：一个用于输入 JSON Schema，另一个用于输入待验证的 JSON 文件。现在我将示例 4-9 的 schema 和示例 4-10 的 JSON 分别粘贴到两个区域，会看到如下错误信息：

- Field: data.name, Error: has longer length than allowed , Value: "Fluffy the greatest cat in the whole wide world"
- Field: data.age, Error: is less than minimum, Value: -2

JSON Schema Validator 这个网站也包含相同功能的两个区域。再一次将示例 4-9 和示例 4-10 输入，将看到如下的错误信息。此外，JSON 中出错的那

一行的行号也会变成红色，以告诉我们具体哪一行出了错误。

- Message: String 'Fluffy the greatest cat in the whole wide world' exceeds maximum length of 20, Schema Path: #/properties/name/maxLength
- Message: Integer -2 is less than minimum value of 0, Schema Path: #/properties/age/minimum

除了提示出错行以外，JSON Schema Validator 还会告诉我们导致验证失败的相关规范的具体路径。这两个验证器虽然提示出错的方式稍有不同，但指出的错误是相同的。

4.3 专业术语和概念

本章涵盖了以下专业术语。

- JSON Schema
  数据交换中的一种虚拟的“合同”。

我们还讨论了以下重要概念。

- JSON 验证器负责验证语法错误，JSON Schema 负责提供一致性检验。
- JSON Schema 是数据接收方的第一道防线，也是数据发送方节约时间、保证数据正确的好工具。
- JSON Schema 可以解决下列有关一致性验证的问题。
  - *值的数据类型是否正确？*
    可以具体规定一个值是数字、字符串等类型。
  - *是否包含所需要的数据？*
    可以具体规定哪些数据是需要的，哪些是不需要的。
  - *值的形式是不是我需要的？*
    可以指定范围、最小值和最大值。

第 5 章

JSON中的安全问题

JSON 本身不会构成什么威胁，毕竟它只是一种数据交换格式。它不过是一种数据文件，或者说数据流。真正会产生安全问题的是 JSON 的使用。本章重点讨论在 Web 中使用 JSON 时最常见的两个安全问题：跨站请求伪造和跨站脚本攻击。

在讨论安全问题和本书后面的内容前，有必要了解一下客户端和服务端的关系。如果你还不了解这些概念，那么请跟我简单了解一下。

5.1　客户端和服务端的关系

和往常一样，我又一次来到 Pierre’s Fine Dining 餐厅享用晚餐。我坐在桌边，看着桌上被折叠成天鹅形状的餐巾，等待着服务员过来点餐。不一会儿，一个穿着 T 恤和长裤的帅小伙走过来说：“我叫托马斯，是这里的服务员。”很快，他认出了我，把声音压低了些说道：“顺便说一句，您可是我最棒的顾客之一。”他睫毛闪动，继续说：“这是我们的特供菜单。”

看完特供菜单后，我点了菜，托马斯很快就把菜上好了。我美美地用完餐，并为这些精美可口的餐点付了 200 美元。这是我在 Pierre’s Fine Dining 餐厅度过的又一个美妙的晚上。在这里，托马斯是服务员，我是客户。

互联网浏览器和网站之间的关系就像我和餐厅的关系一样。这一关系包含了大量的请求和响应。我的请求是一份晚餐，厨房的响应则是将我点的菜做出来并送到我的面前。

举个例子，假如你正在喜欢的网站上看可爱小猫的图片，你电脑上的互联网浏览器就是客户端，而运行着可爱小猫图片的网站的电脑就是服务端。你的浏览器通过互联网将请求发送给小猫图片网站的服务器，服务器接着就会把对应的页面作为响应发送给你。接下来你的浏览器就会将页面在屏幕上渲染出来。

在这一关系中，我们称图片网站返回的即将被浏览器处理的响应为客户端代码。前面提到的饭店的例子中，响应就是送来的晚餐，而餐桌就像网络浏览器。晚餐放到桌子上，然后我就能享用了。

我们将页面响应传递过来之前所发生的事（主要是页面的创建）称为服务端代码。饭店的例子中，服务端代码就是在厨房中发生的事情。我不会到厨房去，也不会去看他们把饭做出来都经历了什么过程。这个例子和实际情况唯一的不同点是，服务器不是那个在餐桌和厨房间来回跑的服务员。厨房像是服务器，服务员则像是互联网。

网站不会把它在“厨房”中做的事公开。网站可能使用 PHP、ASP.NET 或其他编程语言。无论“厨房”里发生什么，对我的浏览器都不会产生什么影响，只要它交付正确的客户端代码即可。

具体来讲，我在客户端收到的响应就是 HTML、CSS 和 JavaScript 代码。就跟能看到餐桌上有什么菜一样，只需按下键盘上的 F12，打开开发者工具，便可看到页面包含的所有的 HTML、CSS 和 JavaScript 代码。我甚至可以看到那些烦人的小猫跳舞的 flash 效果是怎么用 JavaScript 来实现的。

客户端就是发生在用户浏览器中的一切，而服务端则是发生在运行网站的服务器中的一切。当我们提起客户端代码时，通常指的是 JavaScript、HTML 或 CSS。当我们提到服务端代码时，常常是指一些服务端语言，如 ASP.NET、Ruby on Rails 或 Java。

现在可以讨论安全问题这个重要的话题了。

5.2 跨站请求伪造

跨站请求伪造，即 CSRF（cross-site request forgery，读作 sea-surf），是一种利用站点对用户浏览器信任而发起攻击的方式。CSRF 漏洞已经存在了很长时间，远比 JSON 出现得早。

接下来看一个利用 JSON 进行 CSRF 攻击的例子。

你登录了一个银行的网站，这个网站有一个关于你的敏感信息的 JSON URL（示例 5-1）。

示例 5-1：JSON 中保存的你的敏感信息

```
[
    {
        "user": "bobbarker"
    },
    {
        "phone": "555-555-5555"
    }
]
```

你可能会说："嘿，你的 JSON 最外面没加花括号！"其实这里的 JSON 格式是合法的。不过有些合法的 JSON 十分危险，因为它们也是可以执行的 JavaScript 脚本。本例这种情况也被称为*顶层 JSON 数组*。

本例中的站点使用会话 cookie 验证，以确保信息是传送给像你一样已经注册且处于登录状态的用户。

而示例中的黑客在发现银行网站上保存敏感信息的 JSON 的 URL 后，会把它放到指向自己站点的 `<script>` 标签中。如果不确定 `<script>` 标签是什么，可以在大多数现代的 Web 页面中找到它。这里所说的 Web 页面就是所谓的背后的代码，本质上就是一个 HTML 文档。在网页中单击右键，选择审查元素，就会看到一个或多个类似于示例 5-2 中所示的标签。

示例 5-2：`<script>` 标签示例（"src"属性指该脚本位于哪里）

```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
```

浏览器对于不同域名（http://domainone.com 和 http://domaintwo.com 就是两个不同的域名）的站点之间进行资源分享有一定的限制规则。黑客使用

<script> 标签来绕开这些规则（示例 5-3）。有时我们会使用其他站点提供的脚本，所以 <script> 标签不受规则的限制。使用顶层级别数组的 JSON 是合法的 JavaScript 代码，所以他就得到了我们的数据。

示例 5-3：黑客站点的 <script> 标签可能是这个样子的

```
<script src="https://www.yourspecialbank.com/user.json"></script>
```

这样的攻击能够实现，很关键的一点就是利用了你和网站间的凭证。没有你的凭证，<script> 标签中的链接是不会返回任何东西的。那个链接中保存的并不仅仅是你的敏感信息。那是一个动态的链接，会根据登录用户的不同而显示不同的敏感信息。当你在网站登录时，便初始化了一个凭证，网站以此来确定你的身份。[2]

CSRF 攻击就利用了这一信任关系。黑客为了利用这一信任，需要你在已经登录银行的情况下访问他含有危险的 <script> 标签的网站。要实现这点，他会发送大量的伪造邮件，内容大致是“银行提醒您：重要消息”。这类邮件会伪造得和你的银行（或它们想攻击的网站）发来的邮件一模一样。如果接收者不好好查看一下发件人，或不仔细查看一下链接是否是指向他们可以信任的网站，就很可能会点进去。

举例来说，假如有一天你生病了，精神状态不太好，不小心点击了这个链接。如果这时你没有退出银行账号，会话就仍然存在。此时你和银行之间处于信任的关系中。一旦你进入了坏人的网站，意识到这个站点很奇怪，就关闭了。不过这时已经晚了。黑客已然获取到了敏感的 JSON 数据并发送到了自己服务器上保存起来。

那么银行及其开发者应如何阻止 CSRF 攻击呢？

首先，银行应该将数组作为一个值存入 JSON 对象。这样数组将不再是合法的 JavaScript。见示例 5-4。

注 2：关于示例中攻击的具体介绍，可以参考：http//www.cnblogs.com/hyddd/archive/2009/07/02/1515768.html 和 http://drops.wooyun.org/papers/42。——译者注

示例 5-4：将数组存放到对象之中，使其成为非法的 JavaScript，这样就不会被 <script> 标签加载

```
{
    "info": [
        {
            "user": "bobbarker"
        },
        {
            "phone": "555-555-5555"
        }
    ]
}
```

下一步，银行应仅允许 POST 请求获取数据，禁止使用 GET 请求，这样黑客便无法使用他自己的 URL 中的链接了。GET 和 POST 是 HTTP 提供的用于与服务器交换数据的两种方法。GET 用于请求数据，得到响应。POST 用于提交数据，得到响应。如果服务器端允许 GET 请求，就可以直接通过浏览器或 <script> 标签链接到它。但 POST 则不可以直接被链接到。一旦不能使用 <script> 标签，黑客就会受到资源共享策略的限制，从而无法通过利用银行对客户端的信任进行欺骗行为。

当然，这并不意味着通过 HTTP 进行的 JSON 数据交换都应通过 POST 来进行。对于是否允许 GET 方法请求页面或资源，应该考虑的问题是：该页面是否需要通过 URL 或 <script> 标签来加载？如果回答是否定的，那么应该禁用 GET 方法，来阻止他人通过 URL 和 <script> 标签获取数据。

用户的*敏感数据*也是引发窃取行为的关键。如果 JSON 中包含的数据仅仅是一些鸟的种类的清单，黑客也不会专门建一个网站来窃取它。尽管如此，为了防患于未然，养成良好的习惯非常重要。不在 JSON 中使用顶级数组，且不要贪图使用 GET 代替 POST 的便利，这样你就不会写出那些满是漏洞、让用户感到愤怒的代码。

5.3　注入攻击

注入攻击包含许多种形式与格式。不过，它们都是利用系统本身的漏洞来实现的。CSRF 攻击仅包括利用信任机制进行的攻击。注入攻击则主要通过向网站注入恶意代码来实现。

5.3.1 跨站脚本攻击

跨站脚本攻击（cross-site scripting，XSS）是注入攻击的一种。在使用JSON时常见的安全漏洞通常发生在JavaScript从服务器获取到一段JSON字符串并将其转化为JavaScript对象时。

要知道，JSON本身仅仅是文本。在编程中，如果想要对代表对象的文本进行操作，首先要将它转换成对象并装入内存中。这样，它才能被操作、观察，并在程序逻辑中使用。

在JavaScript中，可以使用 `eval()` 函数来进行这一操作。该函数获取一段字符串，并对其进行编译与执行。

下面的示例5-5使用 `eval()` 函数来将animal/cat对象放入内存中。之后就可以在代码中使用该对象的属性了。代码第3行的 `alert` 函数会在浏览器端弹出内容为“cat”的框。

示例5-5：获取对象的属性

```
var jsonString = '{"animal":"cat"}';
var myObject = eval("(" + jsonString + ")");
alert(myObject.animal);
```

示例5-5中的代码相对来说没有什么危险，因为JSON是直接位于代码中的。不过在一些情况下，我们的JSON是从别的服务器上获取的。该服务器通常会是你无权控制的第三方服务器。例如，当你从Facebook的服务器上请求JSON数据时，你没办法控制你获得的JSON是什么样子的。如果服务器本身或发来的JSON数据被人劫持，那么很可能就会运行恶意代码。

`eval()` 函数的问题是，它会将传入的字符串无差别地编译执行。如果从第三方服务器中获取的JSON被替换为了恶意脚本，那么我的站点就会无辜地蒙冤，在访问者的浏览器中编译执行恶意代码。

在示例5-6中，我用一些JavaScript代码来替换原本的JSON字符串。当代码执行时，`eval()` 函数就会执行 `alert()` 函数并弹出“this is bad”。

示例5-6：被 `eval()` 函数执行的 `alert`

```
var jsonString = "alert('this is bad')";
var myObject = eval("(" + jsonString + ")");
```

```
alert(myObject.animal);
```

随着 JSON 本身的不断发展，这一漏洞已得到公认。JSON.parse() 函数就是用来处理这一问题的。该函数仅会解析 JSON，并不会执行脚本。在示例 5-7 中，我们使用 JSON.parse() 来代替 eval()。

示例 5-7：使用 JSON.parse() 代替 eval()

```
var jsonString = '{"animal":"cat"}';
var myObject = JSON.parse(jsonString);
alert(myObject.animal);
```

Web 开发中还有一个和安全同等重要的问题，那就是跨浏览器兼容。随着 Web 的发展，网站的管理者必须要对支持哪些浏览器和使用浏览器的哪些版本有所取舍。实际情况中，肯定会有不少用户不升级他们的浏览器，或是使用一个不支持新标准的老式浏览器。

JSON.parse() 函数更加安全，目前已经被全部主流浏览器的最新版本所支持。然而也有一些仍占有着一小部分用户的老式浏览器不支持该函数。通常情况下，可以通过一些优雅的方式来处理这一问题。比如，可以将这一错误捕获，并弹出一条形如“请升级你的浏览器至最新版本”的消息，来避免让用户看到满是错误的页面。

5.3.2 安全漏洞：决策上的失误

在本章的开始，我曾说过“JSON 本身不会构成什么威胁”。虽然这话本身没什么错误，不过有时候，危险也会直接包含在合法的 JSON 数据之中。

让我们先来看一些很规矩的 JSON 数据（示例 5-8）。

示例 5-8：很规矩的 JSON

```
{
    "message": "hello, world!"
}
```

假如我现在有一个网站，其页面上显示的信息都保存在数据库中，页面上的内容是供用户读取的。由于我从未听说过有关窃取的问题，因此我的站点允许一个用户向另一个用户发送可以包含任何内容的消息。在消息界面，我从服务端请求 JSON 格式的消息并在客户端使用 eval() 函数来将 JSON

转换成可以使用的 JavaScript 对象。然后我将该 JavaScript 对象的 message 属性的值直接显示在 HTML 中。

示例 5-9 展示了一种不是那么规矩的 JSON。

示例 5-9：不那么规矩的 JSON

```
{
    "message": "<div onmouseover=\"alert('gotcha!')\">hover here.</div>"
}
```

示例中这段不怎么规矩的 JSON 中包含了 JavaScript 脚本。当这段 JavaScript 处于 JSON 名称 – 值对的范围中时，它就是一段字符串文本。这是一段合法的 JSON 数据，而且就 JSON 本身而言，它也确实没什么问题。

有问题的是示例中的这段 JavaScript，当它在网站的消息页面中输出时，就会构成威胁。示例中的这段“不规矩的 JSON”会在用户每次将鼠标移动至屏幕上的消息时弹出内容为“gotcha!”的警告框。而这个问题可能产生的后果可不只警告框那么简单。黑客可以通过该脚本获取你在这一页面上的所有私人信息，并发送到他自己的网站上保存。

如何阻止这种情况呢？一方面，可以采取一些手段使得消息中不包含 HTML。可以在客户端和服务端都加上这一认证。此外，还可以将消息中所有的 HTML 字符进行转码，这样的话，诸如 `<div>` 这样的标签就会被转换成 <div>，然后插入页面（<div> 将不会是合法的 HTML）。所有这些方法都可以在网站中用具体的代码在客户端和服务端实现。

许多注入攻击之所以成功，都是由于网站在架构过程中忽视了一个问题：黑客如何利用这一点？允许在 JSON 中使用 HTML 以及直接将值插入页面都是一些幼稚的决策。抵御注入攻击的关键是要找出可能的注入点，并加入一些额外的步骤（有时可能会很麻烦）来加以防范。

5.4　专业术语和概念

本章涵盖了以下专业术语。

- 服务端（Web 开发中的）
  当网页或资源被请求时，在服务器上执行的一系列操作。服务器为互联网浏览器提供其处理和 / 或加载的响应。

- 客服端（Web 开发中的）
  当浏览器所请求的页面加载完毕时执行的一系列操作。通常是指 HTML、CSS 和 JavaScript。

- 跨站请求伪造（CSRF）
  指利用站点对用户浏览器的信任进行的攻击。

- 顶层 JSON 数组
  存在于 JSON 名称－值对之外的位于文档最顶层的 JSON 数组。

- 注入攻击
  依赖于将数据注入到 Web 应用程序以方便恶意数据执行或编译的攻击。

- JSON 跨站脚本攻击
  通过截取或将站点中所使用的第三方代码更换为恶意脚本，来对站点进行的一种注入攻击。

我们还讨论了以下重要概念。

- JSON 本身不构成什么威胁，它只是文本。
- 在定位 JSON 安全问题时，应该记住以下三件事。
  - 不要使用顶级数组。顶级数组是合法的 JavaScript 脚本，它们可以用 `<script>` 标签链接并使用。
  - 对于不想公开的资源，仅允许使用 HTTP `POST` 方法请求，而不是 `GET` 方法。`GET` 方法可以通过 URL 来请求，甚至可以放在 `<script>` 标签中。
  - 使用 `JSON.parse()` 来代替 `eval()`。`eval()` 函数会将传入的字符串编译并执行，这会让你的代码易被攻击。应仅使用 `JSON.parse()` 来解析 JSON 数据。
- 安全漏洞通常是由于开发人员没有考虑“黑客如何利用这一点”这一问题所造成的。

第 6 章

JavaScript中的XMLHttpRequest与Web API

JavaScript 中的 `XMLHttpRequest` 与 Web API 等概念听上去好像很难，但实际上并没有想象中那么复杂。它仅仅是一种简单的客户端与服务端的关系。JavaScript 中的 `XMLHttpRequest` 负责在客户端发起请求，而 Web API 负责在服务端返回响应。

在上一章中，我曾用餐厅的例子来说明客户端和服务端的关系。服务端就像是厨房，客户端就像是来用餐的人。而这一章关注的就是其中一类厨房，看看它是如何运作的。

毋庸置疑，不同餐厅的运作方式千差万别。在有些餐厅，你可以驱车到用餐窗口点餐。有些餐厅对公众开放，而有的则只是某个大公司的内部食堂。

我们平时接触最多的一种客户端－服务端关系就是上网。通常情况下，我们会认为自己是在网上不断地浏览和探索，而实际上一直都是坐在椅子上盯着电脑屏幕。浏览器哪都没有去。它就像是餐厅的桌子，老老实实待着，发送请求，接收响应。一旦服务端回应了请求，便去着手回应其他的请求了。

互联网浏览器发送的是对某项资源的请求。上网时，要么点击一个指向某个 URL 的链接，要么直接在浏览器中输入 URL。URL 的全称是通用资源标识符。我们在浏览器中使用的 URL 通常指向 HTML 资源，它能让我们看到我们的网站，比如前面提到的可爱小猫图片网站。在这种情况下，我们所请求的资源的内容类型为 text/html。

另一种与我们并不直接相关的客户端－服务端关系是 Web API。Web API 的服务内容和普通网站差不多，但是它并不是用来拿给人看的。你可以把它想象成为代码服务的餐厅，毕竟大多数发往这类服务器的请求都是由代码发起的。

程序代码不会像我们一样请求小猫的图片来看。通常情况下，它所需要的是获取数据。在这一章，我们将着眼于一种请求 JSON 资源（一种内容类型为 application/json 的资源）的客户端，以及为这类顾客服务的餐厅：Web API。

现在看看什么是 Web API，以及 JSON 在其中所扮演的角色。

6.1 Web API

和人不同，代码没有一双能够读书或看图的眼睛。它只能以一种它能读取（解析）的格式来查看“某样东西”。这也就是数据交换格式（如 JSON，见示例 6-1）的用武之地。

示例 6-1：OpenWeatherMap Web API 所提供的 JSON 格式的天气数据

```
{
    "dt": 1433383200,
    "temp": {
        "day": 293.5,
        "min": 293.5,
        "max": 293.5,
        "night": 293.5,
        "eve": 293.5,
        "morn": 293.5
    },
    "pressure": 1015.06,
    "humidity": 98,
    "weather": [
        {
            "id": 802,
```

```
                "main": "Clouds",
                "description": "scattered clouds",
                "icon": "03n"
            }
        ],
        "speed": 2.86,
        "deg": 134,
        "clouds": 44
    }
```

这段 JSON 格式的天气数据，可以被任何能够解析 JSON 的代码所使用。JSON 资源可以通过一个 URL 来请求（上面的代码示例是一份完整 JSON 文档的一部分）：

http://api.openweathermap.org/data/2.5/forecast/daily?lat=35&lon=139&cnt=10&mode=json

尽管许多 Web API 就像 OpenWeatherMap 一样只有“读”的功能，但也有许多诸如 PayPal API 这样的 API 还具有交互性。Web API 是通过 HTTP 服务进行交互的一组指令和标准。这些交互可以包括创建、读取、更新、删除（CRUD）等操作，且 Web API 都会有一份说明，概述如何使用这些指令和标准。

例如，根据 PayPal API 的说明，我将通过向如下 URL 发送 JSON 数据的方式，使用 PayPal API 创建一张新的发票：

https://api.sandbox.paypal.com/v1/invoicing/invoices

示例 6-2 中，我将把一段代表一张发票的 JSON 数据作为请求发送给 PayPal API。

示例 6-2：PayPal API 的一张 JSON 发票

```
    {
        "merchant_info": {
            "email": "bob@bob.com",
            "first_name": "Bob",
            "last_name": "Bobberson",
            "business_name": "Bob Equipment, LLC",
            "phone": {
                "country_code": "001",
                "national_number": "5555555555"
```

```
            },
            "address": {
                "line1": "123 Fake St.",
                "city": "Somewhere",
                "state": "OR",
                "postal_code": "97520",
                "country_code": "US"
            }
        },
        "billing_info": [
            {
                "email": "someguy@someguy.com"
            }
        ],
        "items": [
            {
                "name": "Widgets",
                "quantity": 20,
                "unit_price": {
                    "currency": "USD",
                    "value": 89
                }
            }
        ],
        "note": "Special Widgets Order!",
        "payment_term": {
            "term_type": "NET_45"
        },
        "shipping_info": {
            "first_name": "Some",
            "last_name": "Guy",
            "business_name": "Not applicable",
            "address": {
                "line1": "456 Real Fake Dr",
                "city": "Some Place",
                "state": "OR",
                "postal_code": "97501",
                "country_code": "US"
            }
        }
    }
```

有了 PayPal API，一旦发票被创建，就可以请求（读取）、更新以及删除该发票了。

JavaScript 在幕后进行的这些操作，如请求天气数据，称为异步操作。异步操作通常指那些发生在幕后的、不会中断主进程的操作。

在 JavaScript 的异步操作中，“主进程”指 Web 浏览器的显示进程。例如，一个新闻页面可能会包含一个实时显示天气数据的侧边栏。在阅读新闻时，后台的代码会每隔 60 秒异步更新显示的天气数据。而这一操作并不需要刷新页面，也不会在阅读文章时对页面滚动产生什么影响。页面中唯一发生变化的只是包含天气信息的侧边栏。

这里提到的 JavaScript 中的异步（后台）操作被称为 AJAX。AJAX 的全称是 Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）。当在幕后请求 JSON 数据时，这从技术上讲可以被称为异步的 JavaScript 和 JSON（AJAJ）。然而，“AJAX”这个专有名词使用的时间已经很久了，所以很多时候，与其说它是一个首字母缩写，不如说是用来描述 JavaScript 中的任何一种异步操作。在本书以及其他许多地方，经常能看到使用 AJAX 来描述异步的 JavaScript 和 JSON。这并没有什么错误。

接下来看看如何用 JSON 和 JavaScript 中的 `XMLHttpRequest` 实现 AJAX。

6.2　JavaScript中的XMLHttpRequest对象

尽管 JavaScript 的 `XMLHttpRequest` 对象听上去和 XML 有关，但实际上我们使用它来发起 HTTP 请求。在它以包含 XML 的名字命名时，XML 是在发起这类请求时最常用的数据交换格式。然而，`XMLHttpRequest` 并不仅限于使用 XML。让我们别管它的名字，安心地使用它来请求 JSON 吧。

前面提到了主动去获取资源的代码。接下来讲的 JavaScript `XMLHttpRequest` 就是这样一类代码。尽管实际上代码不会像狗追球那样在网络空间跑来跑去，不过当讨论 HTTP 请求时，还是常常会用到这个词。代码不会动，只会静静地在原地发出请求，等待资源被送达。

回想前面餐厅的那个例子，我们将 JavaScript 代码视为坐在餐桌旁的顾客。它想要做的就是请求 JSON 这道“餐点”，为了实现这一目的，它不需要去厨房，也不必在餐厅里嚷嚷。餐厅里有一份协议，该协议使得你只需要的东西交给服务员，由他转交给厨房就可以了。

同样，也有一些协议允许代码向距离很远的服务器发送请求。那些允许我们通过访问站点进行数据交流的基础是基于*超文本传输协议*（HTTP）的。

在浏览器中输入 http://www.cutelittlekittens.com 时，就是在使用 HTTP 协议去请求资源。小猫图片网站中的资源就是包含小猫图片的 HTML 页面。请求 JSON 资源的代码也同样使用 HTTP 协议。

JavaScript 中，使用这种协议来发送这类请求的代码就是 `XMLHttpRequest`。JavaScript 是一种面向对象的语言，而 `XMLHttpRequest` 就是一类对象。当使用 `new XmlHttpRequest()`（示例 6-3）语法，并将其返回值赋值给一个变量时，它就具有了从某一地址请求资源的功能。

示例 6-3：JavaScript 中的 `XMLHttpRequest` 对象

```
var myXmlHttpRequest = new XMLHttpRequest();
```

`XMLHttpRequest` 是一个对象，示例 6-3 就展示了如何创建一个 `XMLHttpRequest` 对象。即便没学过 JavaScript，经过这么多章的学习，你学到的关于 JSON 的知识肯定可以帮你了解 JavaScript 对象了。毕竟，JSON 基于 JavaScript 对象的字面量表示法。

我们说过，JavaScript 对象具有属性。这些“属性”就是名称 – 值对。`XMLHttpRequest` 对象所拥有的属性有 `onreadystatechange`、`readyState`、`response`、`responseText`、`responseType`、`responseXML`、`status`、`statusText`、`timeout`、`ontimeout`、`upload` 以及 `withCredentials`。这些属性在命名方式上有些混乱。有些使用驼峰命名方式，有些使用小写字母。当你将来自己写 `XMLHttpRequest` 代码时，它们可能会引起些小麻烦，毕竟你不光要记住它们的名字，还要记住它们的写法。

编程中典型的对象和 JSON 对象的一点很关键的区别是 JSON 对象中不包含可执行的指令。由于 JSON 是用于数据交换的，所以它只包含属性。而编程中典型的对象还会包含函数（或方法）。例如，我用一些 JSON 的名称 – 值对来形容我的鞋，如：`"color": "pink"`，`"brand": "crocs"`。但我不能通过 JSON 来赋予它生命，比如调用一个类似 `shoe.walk()` 的函数。

我们主要关注 `XMLHttpRequest` 中的以下这些可用的函数：

- `open`（`method`，`url`，`async`（可选），`user`（可选），`password`（可选））
- `send()`

以及下面这些属性：

- `onreadystatechange`

  可以在代码中给它赋值为一个函数

- `readyState`

  返回一个 0~4 的值，用来表示状态码

- `status`

  返回 HTTP 状态码（如 200 表示请求成功）

- `responseText`

  当请求成功时，该属性会包含作为文本的响应体（如我们请求的 JSON）

如果你没学过 JavaScript，请记住这句话：属性的值可以是一个函数。因为 JavaScript 中的函数也是一类对象。对象是一类数据，因此它可以被赋值给一个变量（属性）、修改和传递。在编程中，这种情况称为“函数是一等公民”。`onreadystatechange` 的值应该是一个函数。

示例 6-4 中创建了一个新的 `XMLHttpRequest` 对象，并让它从 OpenWeather-Map API 获取 JSON 数据。

示例 6-4：一个新的 `XMLHttpRequest` 对象

```
var myXMLHttpRequest = new XMLHttpRequest();
var url = "http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139";

myXMLHttpRequest.onreadystatechange = function() {
    if (myXMLHttpRequest.readyState === 4 && myXMLHttpRequest.
        status === 200) {
        var myObject = JSON.parse(myXMLHttpRequest.responseText);
        var myJSON = JSON.stringify(myObject);
    }
}
myXMLHttpRequest.open("GET", url, true);
myXMLHttpRequest.send();
```

在本例的第二行代码中，我创建了一个保存着 JSON 资源的 URL 的字符串。然后我将一个函数赋值给 `myXMLHttpRequest` 的 `onreadystatechange` 属性。该函数会在每次 `readyState` 属性发生变化时执行。在这一函数中，我会判断 `readyState` 值是否为 4（表示“完成”），以及 HTTP 状态码是不是

200（表示“成功”）。如果这两个条件都返回 `true`，就将 JSON 文本解析成 JSON 对象。

在将一个对象转成 JSON 文本，或是将 JSON 文本转成对象时，很可能会接触这两个专业术语：*序列化*和*反序列化*。序列化是将对象转换成文本的过程，反序列化是将文本转换成对象的过程。

示例 6-5 中使用 JavaScript 中的 `JSON.parse()` 进行反序列化操作。响应返回的 JSON 以文本的形式存储在 `responseText` 中。当它被 `JSON.parse()` 解析后，就不是 JSON 了，而是 JavaScript 对象。

示例 6-5：反序列化

```
var myJSON = JSON.parse(myXMLHttpRequest.responseText);
```

由于 JSON 一开始还不是对象，所以使用 `JSON.parse()` 进行反序列化是有必要的。请记住，JSON 意思为 JavaScript 对象*表示法*。当它以 JSON 形式存在时，字面上代表的是以文本形式表示的一个对象。为了让 JSON 变为真正的对象，需要进行反序列化操作。在 JavaScript 中，也可以通过 `JSON.stringify()` 对 JSON 进行序列化。

示例 6-6 中，`myObject` 变量得到了一个反序列化后的 JSON。现在它是一个对象了。`myJSON` 变量得到了一个序列化后的 JSON。现在它是一段 JSON 文本。

示例 6-6：对象的序列化和反序列化

```
// JSON响应的反序列化
var myObject = JSON.parse(myXMLHttpRequest.responseText);

// 对象的序列化
var myJSON = JSON.stringify(myObject);
```

最后，示例中的最后两行代码建立了请求并通过 HTTP 协议发送（示例 6-7）。

示例 6-7：建立一个 JSON 请求并发送

```
myXMLHttpRequest.open("GET", url, true);
myXMLHttpRequest.send();
```

你可能会觉得处理 JSON 响应的代码出现在请求发送前有些奇怪。甚至是在建立和发送请求前就说明那段代码的。这里要知道，`onreadystatechange` 属性的函数值是一个时间处理函数。底层的 JavaScript 引擎（不是我的代

码）会在每次就绪状态（ready state）改变时获取该函数并调用。就绪状态从 0 到 4 分别意味着：

- *0 表示未发送*
  表示 `open()` 函数还没有执行

- *1 表示已发送*
  指 `open()` 函数已执行，但 `send()` 函数还没有执行

- *2 表示接收到头部*
  表示 `send()` 函数已执行且头部和状态码都可以获取了

- *3 表示解析中*
  表示头部已经收到，但响应体正在解析中

- *4 表示完成*
  表示请求完成，包括响应头和响应体的内容都已经接收到了

> 可能这些就绪状态让你有些迷糊。我们会在第 7 章了解到 jQuery 是如何来简化 `XMLHttpRequest` 的。通过使用 jQuery，可以通过更为简单的 `getJSON()` 函数来请求 JSON，并且通过更为人性化的 `done()`、`fail()`、`always()` 等方式来处理各种状态。

我在示例中使用了 OpenWeatherMap API 的 URL。这是一个通过 API 提供天气数据的 Web 服务。该 API 通过不同的 URL 提供 XML 和 JSON 两种格式的天气数据。我在示例中使用 JavaScript 在客户端请求并接收了该 API 的 JSON。然而，出于安全考虑，浏览器对资源共享有一定的限制。同源策略就要求此类后台请求仅可以请求来自同一域名的资源。由于后台请求仅可在浏览器的开发者工具中看见，因此保护了普通用户的安全。

假如前面示例中的代码是在一个名为 http://www.mycoolsite.com 的网站上执行的。由于它是最开始的请求的 URL，所以它就是原始域名。而 API 的 URL 是：

    http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139

共域名 http://api.openweathermap.org 与原始域名 http://www.mycoolsite.com 不匹配。很明显，这违背了同源策略。所以 OpenWeatherMap API 需要使用一些手段来绕过这些限制，才能使后台请求与响应顺利进行。让我们了解一下这些限制和漏洞，好知道后台对公共 API 的请求是如何实现的。

6.3 混乱的关系与共享的规则

我在这里留给了你一些悬念。我展示了一些能够运行的代码，但是它们违背了浏览器的同源策略。它们本来不应该跑通，但是却跑通了，这是因为 OpenWeatherMap API 的开发者们绕过了一些限制。这听上去就像一部糟糕的肥皂剧，两个人始终都不能确定他们的关系。浏览器会不会允许对公共 Web API 的请求呢？让我们拭目以待。

6.3.1 跨域资源共享

有些开发人员可以连续多年通过 JavaScript 的 AJAX 技术向公共 API 发送请求，而不会受到同源策略的影响。这是因为这些公共 API 的开发者在他们的服务器上实现了跨域资源共享（cross-origin resource sharing，CORS）。这些服务器会在响应头额外加上一些带有 `Access-Control-Allow` 前缀的属性（示例 6-8）。

示例 6-8：OpenWeatherMap API 对于请求 http://api.openweathermap.org/data/2.5/weather? lat=35&lon=139 返回的响应头中的 `Access-Control-Allow` 部分

```
Access-Control-Allow-Credentials:true
Access-Control-Allow-Methods:GET, POST
Access-Control-Allow-Origin:*
```

这些头部定义了证书是否可用，哪些 HTTP 方法（`GET`、`POST`、`PUT`、`DELETE`、`HEAD`、`OPTIONS`、`TRACE`、`CONNECT`）是可用的，以及最重要的一点，即允许哪些域名。本例中，此处包含一个星号（`*`）。它表示任意域名都是允许的。

CORS 使得诸如 OpenWeatherMap 这样的站点可以让用户在客户端使用 AJAX 交互的方式获取它们的数据。同时，CORS 还可用于禁止某些域名的访问，来防止有些人用 API 做坏事，如 CSRF。我们在第 5 章的示例中提到

过，黑客企图通过将 JSON 放入 `<script>` 标签来利用银行站点的信任。现在银行有了一个新的安全的方式来防止这种攻击，这个方式就是实现 CORS 并且在其响应头中加入类似示例 6-9 的内容。

示例 6-9：使用 CORS 进行安全防护

```
Access-Control-Allow-Methods:POST
Access-Control-Allow-Origin:http://www.somebank.com
```

本例中，我们仅允许通过 `POST` 方式请求资源。如果有人想通过将 URL 放入 `<script>` 标签来进行请求（使用 `GET` 方式），浏览器会阻止他。同时，由于具体设定了银行站点的 URL，浏览器会禁止除 http://www.somebank.com 以外的站点去获取资源。

使用 CORS 也会遇到一些问题，而且也不是所有的 JSON 数据都是从标准的 Web API 获取的。例如可能你有两个域名不同的站点（http://domainone.com 和 http://domaintwo.com），且想让 http://domainone.com 与 http://domaintwo.com 共享一些 JSON 文件，这就需要使用 JSON-P 了。

6.3.2 JSON-P

JSON-P 指带有 padding 的 JSON。我在第 5 章提到过 `<script>` 标签不受同源策略的影响。JSON-P 就是利用这一点来向不同域名的站点请求 JSON 的。

JSON-P 并没有 CORS 那么理想，它只是一个备选方案（CORS 是更好的方案，同时也是一种标准）。不过有些时候，还是很需要这种不是很规范的解决方案的。当不能使用 CORS 时，仍需要通过某种关系来实现 http://domainone.com 与 http://domaintwo.com 之间的交流。

JSON-P 中的“padding”（内联）非常简单，就是将 JavaScript 加入 JSON 文档。见示例 6-10。

示例 6-10：JSON-P

```
getTheAnimal(
    {
        "animal": "cat"
    }
);
```

内联于 JSON 文档的 JavaScript 调用了一个函数，函数参数是 JSON。函数参数提供了一种将数据传递给函数的方式。例如，如果需要一个函数来实现两数相加的功能，首先就需要一个能够将两个数传递给函数的方式。

该函数是在客户端的 JavaScript 代码中定义的（示例 6-11）。

示例 6-11：在 JavaScript 中声明的函数

```
function getTheAnimal(data) {
    var myAnimal = data.animal; // will be "cat"
}
```

当在 JavaScript 中声明该函数之后，需要进行一些设置。这部分就体现出了 JSON-P 是如何利用 `<script>` 标签不受同源策略影响这一点的。见示例 6-12。

示例 6-12：创建 `<script>` 标签，并将其动态添加到 HTML 文档的 `<head>` 标签中

```
var script = document.createElement("script");
script.type = "text/javascript";
script.src = "http://notarealdomain.com/animal.json";
document.getElementsByTagName('head')[0].appendChild(script);
```

服务端也需要对 JSON-P 提供一定的支持，它应允许用户自定义函数的名字。它通常是作为 URL 中 `queryString` 的参数传递的。见示例 6-13。

示例 6-13：通过 `queryString` 告知服务器函数的名字

```
script.src = "http://notarealdomain.com/animal.
json?callback=getThing";
```

服务端会根据 `callback` 参数的值来动态地为在 JSON 中内联的函数命名。见示例 6-14。

示例 6-14：JSON-P 中动态命名的函数

```
getThing(
    {
        "animal": "cat"
    }
);
```

JSON-P 还需要服务端的不少支持，因为 JSON 资源必须包含 JavaScript 内联。不管是使用 CORS 还是 JSON-P，都离不开服务端的支持。因此，客户

端跨域的 XMLHttpRequest 需要服务端的支持来保证 JSON 资源请求成功。

6.4 专业术语和概念

本章涵盖了以下专业术语。

- Web API
  通过 HTTP 与服务进行交互的一系列指令与标准。
- XMLHttpRequest
  一种 JavaScript 对象，无需刷新页面即可从一个 URL 获取数据，常用于 AJAX 编程。
- 超文本传输协议（HTTP）
  万维网使用的交换数据的基本协议。
- 序列化
  将对象转化为文本的操作。
- 反序列化
  将序列化的文本转化为对象的操作。
- 同源策略
  出于安全的考虑，浏览器仅会请求同一个域的脚本。
- 跨域资源共享（CORS）
  通过设置响应头，使得跨域请求资源（如 JSON 文档）可以成功。
- JSON-P
  使用 <script> 标签，绕过同源策略的限制，以从不同域名的服务器上请求 JSON。

我们还讨论了以下重要概念。

- JavaScript 的 XMLHttpRequest 与 Web API 之间的关系是客户端与服务端之间的关系。
- XMLHttpRequest 并不仅限于 XML，还可以用它来请求 JSON 资源。

- 网站为人服务，Web API 为代码服务，它们都使用 HTTP 协议。
- 同源策略使得 JavaScript 和 JSON 资源进行客户端 – 服务端交流时出现了一些困难。
- 客户端跨域的 `XMLHttpRequest` 需要服务端的支持来保证 JSON 资源请求成功。

第 7 章

JSON与客户端框架

第 6 章探讨了浏览器与 Web API 之间的客户端 – 服务端的关系。本章将深入分析这一关系中的客户端部分，了解客户端的框架为 JSON 提供了哪些支持。

先快速了解一下什么是框架。在我们的生活中，“框架”常被用来形容必要的支持结构，或是底层结构。在计算机中，框架并不是一种底层结构，而是一层位于软件或编程语言之上的为开发者提供支持的结构。

所以，计算机中的框架是一种支持结构，但它和房子的大梁并不一样。如果说 JavaScript 这门语言是一座房屋，JavaScript 框架并不是它的支撑结构。事实上，不使用 JavaScript 框架就可以建造一座结实的房屋。

如果我们是使用 JavaScript 建房子的商人，那么 JavaScript 框架就像是已经准备好水暖和供电的毛坯房。有了它，我们可以更加专注于为厨房选取怎样的水池，而只要把现成的管道与水池相连接，就能获取自来水了。我们也可以专注于安装漂亮的橱柜和花岗岩台面了。本质上讲，JavaScript 框架可以节省时间，让我们更专注于功能的构建。

这种节约时间、专注重点的框架在计算机科学中也被称为抽象化工具。不熟悉抽象化概念的人可能会想到毕加索画中的那些用几何图形构成的奇怪的脸。但是我们所说的抽象化可不是这么复杂的概念，它是在处理复杂系

统时所使用的一种技术。

如果要制作一艘宇宙飞船，但对火箭相关的技术全无了解，该从何着手？大多数人肯定会说“显然我不适合这项工作”，但是如果别无选择呢？你也许会对这项复杂的工作束手无策，觉得这不可能完成，甚至会感到恐惧，把自己锁在屋子里。但是你还有另外一种选择：使用抽象化。

借助抽象化，每次只需要思考问题的一部分即可。可以将建造飞船这一工程分解为必要的几部分，如人类生命供给系统、飞船的发射系统等，然后逐个解决复杂系统的每一个部分，直到完成整个系统。

抽象化工具就是用来帮助进行抽象化工作的。JavaScript 框架中包含一些预定义的库，这些库已经解决了构建系统中的一些复杂问题，能让我们更方便地进行抽象化。虽说没有框架的话，我们可能造不出一艘飞船，但如果框架中包含一艘可以发射到太空的飞船的话，就能造出来，这样我们就可以将精力投入到人类生命供给系统的构建中了。

如今，有许多（>50）种 JavaScript 框架可供选择。通常情况下，他们被称为 JavaScript 库或工具包。它们都在复杂的系统和手头的任务之间创建了一个抽象层。它们大都提供了便于使用的 HTML 文档对象模型（document object model，DOM）操作方法，或是专注于创建成熟的客户端 Web 应用程序。

让我们来了解一些 JavaScript 框架，以及它们与 JSON 的关系。

7.1　jQuery和JSON

jQuery（http://jquery.com/）是一种允许开发者专注于操作 DOM 构建功能的抽象化工具。DOM 为我们和 HTML 页面产生交互提供了便利。在这一模型中，底层的 HTML 可以被当作具有许多可以枚举、获取和操作的字节点的对象来处理。

不使用 jQuery 框架，单纯使用 JavaScript 来对 DOM 进行操作也是可行的。不过，开发人员需要定期对 DOM 执行一些操作，这通常通过写几行代码来实现。例如，将一个 HTML 元素隐藏就是一个很常见的 DOM 操作。比如，

先创建一个按钮，然后隐藏它（示例 7-1）。

示例 7-1：一个在浏览器中显示为“My Button”的按钮

```
<button id="myButton">My Button</button>
```

如示例 7-2 所示，为使用 JavaScript 代码隐藏这个按钮，首先要调用 HTML 的 document 对象的 `getElementById(id)` 方法，然后将对应的 `style.display` 属性值设为 `"none"`。

示例 7-2：用于隐藏“My Button”的 JavaScript 代码；它不会在浏览器中显示

```
document.getElementById("myButton").style.display = "none";
```

若使用 jQuery，可以用不到上例一半长度的代码实现同样的功能（示例 7-3）。

示例 7-3：用于隐藏“My Button”的 jQuery 代码；它不会在浏览器中显示

```
$("#myButton").hide();
```

在实现相同功能的前提下，更短的代码意味着更短的开发时间。为了节约更多的时间，jQuery 还解决了跨浏览器兼容问题。例如，在第 5 章中我用更为安全的 `JSON.parse()` 方法代替 `eval()` 方法来解析 JSON。老版本的 IE、火狐和 Chrome 浏览器并不支持 `JSON.parse()` 方法。这意味着当使用 `JSON.parse()` 方法时，对于一小部分没有升级浏览器的用户来说，你的代码是不能运行的。

jQuery 帮你解决了大部分兼容问题，包括上面提到的 `JSON.parse()`。jQuery 有自己的解析 JSON 的方法：用 `jQuery.parseJSON` 函数。示例 7-4 使用 JavaScript 中的 `JSON.parse()` 解析 JSON；示例 7-5 使用 jQuery 内置的函数来解析 JSON。

示例 7-4：使用 JavaScript 中的 `JSON.parse()` 解析 JSON

```
var myAnimal = JSON.parse('{ "animal" : "cat" }');
```

示例 7-5：使用 jQuery 内置的 `jQuery.parseJSON` 来解析 JSON

```
var myAnimal = jQuery.parseJSON('{ "animal" : "cat" }');
```

上面的例子中，使用 jQuery 并没有在代码长度上面体现出优势。但是

jQuery.parseJSON() 函数所起的作用远不是一行代码能说明的。如果研究 jQuery 库中对应的源码，会发现它会先尝试使用原生的 JSON.parse() 函数。如果浏览器不支持，它会使用和 eval() 功能类似的 new Function()。同时，它会对一些不合法的字符进行检测，如果发现其中有注入攻击的威胁，就会抛出错误。

除了 jQuery.parseJSON 以外，还有一个用于通过 HTTP 请求 JSON 的函数。回顾第 6 章，在使用 JavaScript 进行 HTTP 请求时，写了不少代码。

示例 7-6： 创建一个新的 XMLHttpRequest 对象并从 OpenWeatherMap API 获取 JSON

```
var myXMLHttpRequest = new XMLHttpRequest();
var url = "http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139";

myXMLHttpRequest.onreadystatechange = function() {
    if (myXMLHttpRequest.readyState === 4 && myXMLHttpRequest.status
        === 200) {
        var myObject = JSON.parse(myXMLHttpRequest.responseText);
        var myJSON = JSON.stringify(myObject);
    }
}
myXMLHttpRequest.open("GET", url, true);
myXMLHttpRequest.send();
```

使用 jQuery 仅需几行代码即可完成对 JSON 数据的请求（示例 7-7）。

示例 7-7： 和前例一样，这段 jQuery 代码创建一个新的 XMLHttpRequest 对象并获取 JSON 数据，还将 JSON 数据反序列化为一个 JavaScript 对象

```
var url = "http://api.openweathermap.org/data/2.5/
weather?lat=35&lon=139";
$.getJSON(url, function(data) {
    // 对天气数据执行一些操作
});
```

jQuery 框架通过缩减请求和解析 JSON 的时间来缩短开发时间，从而对 JSON 提供支持。其中包括了一种支持老式浏览器的解析 JSON 的方法。可以说，jQuery 支持与 JSON 交互。

现在来看看 AngularJS，一个充分利用 JavaScript 对象和 JSON 的优势的库。

7.2 AngularJS

jQuery 框架是为 DOM 操作服务的抽象化工具。AngularJS 框架（http://angularjs.org/）是专注于创建单页应用的抽象化工具。与传统的多页方式不同，单页 Web 应用致力于为用户提供无缝交互的应用体验。

传统多页 Web 应用的方式与用户在客户端和服务端产生的交互是紧密联系在一起的。用户输入或点击 URL，以通过 HTTP 向服务端请求资源。这种用户、客户端、服务端共同参与的 Web 应用，每动一步都需要请求一个新的页面。

在互联网成长为今日能够处理海量多媒体资源的猛兽以前，大多数此类应用都是桌面应用。如果你需要一本数字百科全书，则需要在你的电脑上安装它。这类应用的交互体验多是无缝的。当你在百科全书中搜索内容时，并没有一个在页面变化时跟着变的地址栏。

单页应用的目标就是在你的浏览器中实现同样的无缝浏览体验。这绝大部分是通过 JavaScript 以及我们的好朋友 `XMLHttpRequest` 实现的。当用户仍在当前页面时，幕后的代码已经完成对资源的请求，用户从一个资源跳跃到另一个资源的操作将不再需要通过 URL 或是指向 URL 的链接来进行。

AngularJS 框架这种抽象工具为开发者节约了从零开始构建单页应用的时间。单页 Web 应用是一类复杂的系统，而 AngularJS 并不会为其开发者构建一个单页应用。它所提供的，是一个基于模型 – 视图 – 控制器（MVC）架构概念的框架。

AngularJS 实现的 MVC 概念如下所列。

- 模型

  JavaScript 对象即数据模型。

- 视图

  HTML（提供了与模型进行数据绑定的语法）。

- 控制器

  使用 AngularJS 语法来定义和操作与模型和视图间的交互的 JavaScript 文件。

AngularJS 有一定的学习曲线，尤其对于那些长年使用 JavaScript 进行 DOM 操作的人来说更是这样。它需要你转换你对于 DOM 交互的思维。AngularJS 试图将 DOM 操作和业务逻辑解耦。

可以通过对比使用 AngularJS 与不使用 AngularJS 实现相同的功能，来理解何为 DOM 解耦。例如，需要将用户登录页面中的消息“Hello, stranger”（示例 7-8）改为针对登录用户的问候语（示例 7-9）。在 JavaScript 中，需要通过 DOM 操作来实现这一功能。

示例 7-8：用户未登录时的 HTML 消息

```
<h1 id="message">Hello, stranger!</h1>
```

示例 7-9：用户登录后对消息进行修改的 JavaScript 代码

```
if (signedIn) {
    var message = "Hello, " + userName + "!";
    document.getElementById("message").innerHTML = message;
}
```

在 AngularJS 中，由于 HTML 位于视图层，所以应让它随时准备应对数据模型的变化（示例 7-10）。

示例 7-10：使用 AngularJS 属性和数据绑定语法的 HTML 视图

```
<body ng-app="myApp">
    <div id="wrapper" ng-controller="myAppController">
        <h1 id="message">Hello {{ userData.userName }}!</h1>
    </div>
</body>
```

HTML 标签（`<body>` 和 `<div>`）上的 `"ng-app"` 和 `"ng-controller"` 属性设置了一个支持数据绑定的视图。顾名思义，数据绑定就是指借助一系列的占位符将数据“绑定”在页面上。AngularJS 语法要求在数据的占位符的两边加上标识。这些标识由两个左花括号（`{{`）和两个右花括号（`}}`）组成，它们应包裹着占位符（如示例 7-10 中的 `userData.userName`）。

在示例 7-11 中，包含 `userData` 对象的 JavaScript 控制器将被添加到全局作用域中，这会允许视图（HTML）使用该对象进行数据绑定。

示例 7-11：将 `userData` 对象添加到全局作用域

```
angular.module('myApp', [])
```

```
  .controller('myAppController', function($scope) {
    $scope.userData = { userName : "Stranger" };
    if(signedIn)
    {
        $scope.userData = { userName : "Bob" };
    }
  });
```

在 AngularJS 的 JavaScript 控制器文件中，我们不去操作 DOM，而是去操作数据模型。本例中操作的数据模型是 `userData` 对象。现在无论是原生 JavaScript/HTML 的示例还是 Angular 的示例，都实现了同样的功能。在这个简单的示例中，Angular 的版本好像更麻烦一些，跟同样作为抽象化工具的 jQuery 比较似乎更为明显。然而，当你面对的是一个较为复杂的应用，比如开发一个允许用户对其信息执行创建、读取、更新、删除（CRUD）等操作的页面时，使用 Angular 会简化开发过程。

示例 7-11 使用了 JavaScript 对象 `userData` 作为数据模型。AngularJS 通过使用 JavaScript 对象作为 MVC 架构中的模型来使其充分发挥作用。我们已经知道，JSON 是 JavaScript 对象字面量的表示法。那么 JSON 和 AngularJS 该如何协调？

让我们思考一个问题："数据模型是从哪来的？"本例中，我们的数据被硬编码到了控制器中。而回想之前提到的数字百科全书，我们知道了数据其实在别的地方。也就是说，数据应该是保存在某个数据库中的。

如何将数据放入单页应用的数据模型中就是开发者的事了。在 AngularJS 数据模型中，把数据库中的数据放到数据模型中的最常见的方式就是使用 JSON，也就是通过 HTTP 协议来请求 JSON 数据，是一种客户端－服务端的关系。AngularJS 通过其核心服务 `$http` 来使得通过这一协议进行的数据模型检索变得轻松。

示例 7-12 使用 AngularJS 的 `$http` 来从 OpenWeatherMap API 获取天气数据；JSON 数据作为 `weatherData` 对象被添加到全局作用域。示例 7-13 是这一请求的响应。

示例 7-12：从 OpenWeatherMap API 获取天气数据

```
angular.module('myApp', [])
  .controller('myAppController', function($scope, $http) {
```

```
    $http.get('http://api.openweathermap.org/data/2.5/
     weather?lat=35&lon=139').
      success(function(data, status, headers, config) {
            $scope.weatherData = data;
      });
  });
```

示例 7-13：从 OpenWeatherMap API 返回的 JSON 数据


```
{
    "coord": {
        "lon": 139,
        "lat": 35
    },
    "sys": {
        "message": 0.0099,
        "country": "JP",
        "sunrise": 1431805135,
        "sunset": 1431855710
    },
    "weather": [
        {
            "id": 800,
            "main": "Clear",
            "description": "sky is clear",
            "icon": "02n"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 291.116,
        "temp_min": 291.116,
        "temp_max": 291.116,
        "pressure": 1020.61,
        "sea_level": 1028.58,
        "grnd_level": 1020.61,
        "humidity": 95
    },
    "wind": {
        "speed": 1.51,
        "deg": 339.501
    },
    "clouds": {
        "all": 8
    },
    "dt": 1431874405,
    "id": 1851632,
    "name": "Shuzenji",
    "cod": 200
}
```

从 OpenWeatherMap API 返回的 JSON 数据会被 `$http` 反序列化。然后它将作为名为 `weatherData` 的对象被添加到全局作用域，这样就能通过 HTML 视图的插值语法将其作为数据模型进行绑定了。

在示例 7-14 中，我们将描述天气的数据绑定到了视图。天气描述在对象的 `weather` 属性中，`weather` 属性包含一个由对象构成的数组，其中只有一个值。通过 `weather[0]` 来获取这一对象，然后选择 `description` 属性。

示例 7-14：绑定天气描述

```
<body ng-app="myApp">
    <div id="wrapper" ng-controller="myAppController">
        <div>
           {{ weatherData.weather[0].description }}
        </div>
    </div>
</body>
```

写作本章时，天气是“晴朗”。可以编写一个每隔 60 秒便显示当前天气描述的单页应用。HTTP 请求会在后台执行，同时数据模型的更新会自动触发 HTML 模板的更新。不需要初始化请求；所有这些都会在后台进行。或者说，根据 MVC 的概念，我所编写的 JavaScript 代码不需要去改变 DOM。更新数据模型会自动更新视图。

AngularJS 使得 JavaScript 对象和 JSON 在模型－视图－控制器这一架构中大放异彩。jQuery 通过一些简单的函数来对请求和解析 JSON 提供支持。每一种成功的数据交换格式都离不开交换数据组件的支持。透过 Web 的客户端－服务端关系可以看到，无论是 JavaScript 本身还是其强大的抽象化工具，都对 JSON 提供了强有力的支持。在第 8 章和第 9 章，我们将了解这层关系的另一端——客户端——是如何对 JSON 提供支持的。

7.3 专业术语和概念

本章涵盖了以下专业术语。

- *抽象化*

  一种处理复杂系统的技术，主要思想是将一个大问题转换为多个小问题。

- 框架

  一种能够节约时间，以让我们更专注于构建功能的抽象化工具。

- `jQuery.parseJSON()`

  一个 jQuery 的函数，它不仅会调用 `JSON.parse()` 函数，还会兼容那些不支持 `JSON.parse()` 函数的老式浏览器，且通过验证字符来评估字符串，从而避免了可能的安全问题。

- `jQuery.getJSON()`

  `jquery.ajax()` 函数的简写形式，其中包含了将 JSON 解析为 JavaScript 对象的功能。

- 单页 Web 应用

  与传统的多页方式不同的，着力于提供更加无缝的应用体验的网页。

- 模型 – 视图 – 控制器（MVC）

  一种应用架构模式，它将应用分为三部分：模型（数据）、视图（展示）以及控制器（更新模型和视图）

- AngularJS

  一款使用 JavaScript 对象作为数据模型的 JavaScript MVC 框架。

我们还讨论了以下重要概念。

- jQuery 是一款提供了 JSON 请求和解析功能的能够缩短开发时间的抽象化工具。同时它还解决了跨浏览器兼容问题。
- AngularJS MVC 的概念：
  - JSON 是模型（或者说数据模型）
  - HTML 是视图，且提供了与模型进行数据绑定的语法
  - 控制器是使用 AngularJS 语法来定义和操作与模型和视图间的交互的 JavaScript 文件
- AngularJS 使得 JavaScript 对象和 JSON 在 MVC 架构中大放异彩。

第 8 章

JSON与NoSQL

在第 7 章中，我们深入研究了 Web 客户端－服务端关系中的客户端。而本章将探讨一种服务端的数据库。它不仅使用 JSON 文档来存储数据，并且使用 Web API 对外提供接口。

如今我们生活在“信息时代”，随时随地都会有大量的数据和信息向我们涌来。我们可以使用类似维基百科的站点查询某个具体主题的内容，如“naked mole-rat”。关于“naked mole-rat”的信息，肯定是保存在某个地方的，很明显那个地方就是数据库。

如果你曾经使用过数据库，并且对 SQL 很熟悉，那么你用的肯定是*关系型数据库*。关系型数据库是使用表格、行和列来以结构化形式存储数据的。每个表格都会保存有一定意义的数据，比如账户信息。保存着账户信息的表格肯定要与保存着账户地址的表格存在某种联系。一般会有一个键，如某个用于区分账户的列，来给两个表之间建立一种联系。

为了创建、操作和查询这类关系型数据库，我们使用*结构化查询语言*（structured query language，SQL）。使用 SQL，可以在数据库多个表的行和列之间进行查询（示例 8-1）。

示例 8-1：下面展示了一段简单的 SQL 语句，它会从 Account 表中取出 accountID、firstName、lastName 这几列的数据

```
SELECT accountId, firstName, lastName
FROM Account
```

由于关系型数据库中的表之间存在着关联，所以也可以查询多个表格的行和列的数据。

示例 8-2：下面的查询涉及两个表格，它会返回所有账户的 firstName、lastName 数据，以及账户地址中的 street、zip 数据

```
SELECT Account.firstName, Account.lastName, Address.street, Address.zip
FROM Account
JOIN Address
ON Account.accountId = Address.accountId
```

至于 NoSQL，顾名思义，它不是一种关系型数据库。我们不能使用 SQL 从关联在一起的数据库表格的行和列中获取数据。不过，NoSQL 这个名称并不能告诉我们它*是*什么。它只是表明“我找到了一种存储非关系型数据的替代方法”。自然，“替代方法”还会有其更为深入的含义。

NoSQL 数据库也是多种多样的，其差别就相当于自然界中老虎与大象的差别。之所以有这么多种，是因为数据总是有不同的大小、形式以及用途。NoSQL 的创造者意识到了这一点，因此他们发现了这种与传统的关系型模型不同的数据存储与利用方法。

NoSQL 数据库的一个例子是*键值对存储*。键值对存储模型将数据简化为键值对。如果要将英语词典编入数据库，那么用键值对存储非常合适。每一个单词就是一个键，单词对应的定义就是键的值。对于比较简单的数据结构来说，使用这种数据库比使用传统的关系型数据库要合适。

本章将深入探索的 NoSQL 数据库是使用*文档存储*的。这种替代关系型数据库的解决方案是基于文档的概念组织数据的，而不是用表格来组织和关联数据。目前有 20 多种不同的文档存储数据库。它们有的使用 XML 文档，有的使用 JSON 文档。

本章我们来看一种使用 JSON 文档存储数据的文档存储数据库——CouchDB。

8.1 CouchDB数据库

CouchDB（http://couchdb.apache.org/）是一种使用 JSON 文档存储数据的 NoSQL 数据库。

假如我们需要存储顾客的账户信息。如果使用关系型数据库，账户信息需要用多个表进行保存。如果允许一个账户拥有多个地址，那么还需要一个额外的地址表来支持这种一对多的关系。

在关系型数据库中，对于一个账户对应多个地址这种一对多的关系，需要执行联合查询，以便把数据放在一起。此外，如果我要查询一个账户并将多个地址记录集合在一起，其结果将会包含多行。每一行都有重复的账户字段，且每行都有一个不同的地址。因此，虽然数据在数据库中是结构化的，但是进行查询后，它们在行和列中就是扁平化的。

有了 CouchDB，数据就不需要因为它们之间的关系而被分开存储，也不需要在读取时进行重组。其关系会在数据中保存，并且在数据出入数据库时，数据的结构也是保存完好的。

示例 8-3 使用数组中的对象来表示账户和地址间的一对多关系。

示例 8-3：使用 JSON 文档来表示账户

```
{
  "firstName": "Bob",
  "lastName": "Barker",
  "age": 91,
  "addresses": [
    {
      "street": "123 fake st",
      "city": "Somewhere",
      "state": "OR",
      "zip": 97520
    },
    {
      "street": "456 fakey ln",
      "city": "Some Place",
      "state": "CA",
      "zip": 96026
    }
  ]
}
```

由于 CouchDB 使用文档来存储数据，因此当我从数据库中查询一个账户时，得到的直接就是一个结构化的文档。没有必要进行重组。这样既高效又方便。

然而，对于数据存储来说，文档存储也并非万全之策。这种模型可能会在需要多种关系时制造不少麻烦。例如，如果想将一个地址的省、市、区、街道和邮政编码等数据相关联，该怎么办？如果需要这种关系，那最好还是使用关系型模型，毕竟将复杂的关系用一个文档来表示还是很有难度的。

CouchDB 的另一个好处是有利于数据的变化。有些数据会随着时间而发生变化。比如，如果在 1990 年时让用户填写账户信息中的电话号码，可能会让他们填写下面这些字段：

- 家庭电话
- 工作电话
- 传真

现在早就不是 1990 年了，我们需要一个新的字段让用户填写他们的移动电话号码，这时数据就会产生变化。

对于关系型数据库，需要调整地址表的结构来提供一个移动电话的字段。也可以移除账户表的某些字段，并为电话号码新建一张表。

有了 CouchDB，当数据发生变化时就无需修改表的结构了。可以将电话号码表示为 JSON 中的一个由对象组成的数组（示例 8-4），并且允许一个账户保存任意数量的电话号码。

示例 8-4：一个由电话号码对象组成的数组

```
{
  "phoneNumbers": [
    {
      "description": "home phone",
      "number": "555-555-5555"
    },
    {
      "description": "cell phone",
      "number": "123-456-7890"
    },
    {
      "description": "fax",
      "number": "456-789-1011"
```

```
        }
      ]
    }
```

如果账户将来又有了新的号码，只需将其加入数组即可，见示例 8-5。

示例 8-5：在由电话号码对象组成的数组中加入新的“space phone”

```
{
  "phoneNumbers": [
    {
      "description": "home phone",
      "number": "555-555-5555"
    },
    {
      "description": "cell phone",
      "number": "123-456-7890"
    },
    {
      "description": "fax",
      "number": "456-789-1011"
    },
    {
      "description": "space phone",
      "number": "932-932-932"
    }
  ]
}
```

而且，若需要一个新的字段，如“galaxy”，只需把它加在总的记录中即可（见示例 8-6）。无需任何结构改动。

示例 8-6：在账户记录中加入“galaxy”字段

```
{
  "firstName": "Octavia",
  "lastName": "Wilson",
  "age": 26,
  "addresses": [
    {
      "street": "123 fake st",
      "city": "Somewhere",
      "state": "OR",
      "zip": 97520
    }
  ],
  "galaxy": "Milky Way"
}
```

数据作为 JSON 文档在 CouchDB 数据库中出入。那么问题来了，该怎么把数据放进去，又怎么把数据拿出来？ CouchDB 使用基于 HTTP 的 API 实现这一功能。现在来看看 CouchDB API。

8.2 CouchDB API

有了 CouchDB，就可以通过用 HTTP 请求资源的方式来获取数据库中的数据。对于 HTTP，我们是通过 URL 的方式来请求资源的。从 CouchDB API 中请求的资源是一份 JSON 文档（`application/json`）。

如果在本地安装了 CouchDB，并且其中包含一个名为“accounts”的数据库，就可以通过 http://localhost:5984/accounts/ 这一 URL 来从数据库中获取数据。示例 8-7 展示了从“accounts”数据库中返回的 JSON 格式的数据。

示例 8-7：http://localhost:5984/accounts/ 的响应

```
{
  "db_name": "accounts",
  "doc_count": 3,
  "doc_del_count": 0,
  "update_seq": 7,
  "purge_seq": 0,
  "compact_running": false,
  "disk_size": 28773,
  "data_size": 1248,
  "instance_start_time": "1432493477586600",
  "disk_format_version": 6,
  "committed_update_seq": 7
}
```

注意其中的 `"doc_count"` 名称 – 值对。它的含义是数据库中包含多少份文档，例如在我的数据库中是 3 份。可以通过每个文档的唯一标识符来对其进行查询，其中 URL 格式为 http://localhost:5984/accounts/<unique_identifier>。如果事先不知道文档的唯一标识符，可以通过 http://localhost:5984/accounts/_all_docs 这一 URL 来获取行标识符数组。

示例 8-8 展示了 http://localhost:5984/accounts/_all_docs 返回的 JSON 资源，其中包含了由数据库中每份文档的文档标识符所构成的数组。

示例 8-8：文档标识符数组

```
{
  "total_rows": 3,
  "offset": 0,
  "rows": [
      {
          "id": "3636fa3c716f9dd4f7407bd6f7000552",
          "key": "3636fa3c716f9dd4f7407bd6f7000552",
          "value": {
              "rev": "1-8a9527cbfc22e28984dfd3a3e6032635"
          }
      },
      {
          "id": "ddc14efcf71396463f53c0f880001538",
          "key": "ddc14efcf71396463f53c0f880001538",
          "value": {
              "rev": "1-3aef6f6ae7fff90dac3ff5d6c4460ceb"
          }
      },
      {
          "id": "ddc14efcf71396463f53c0f8800019ea",
          "key": "ddc14efcf71396463f53c0f8800019ea",
          "value": {
              "rev": "5-c38761b818edaf9842a63574927b7d38"
          }
       }
    ]
}
```

如果从数组中拿到了第一份文档的标识符（3636fa3c716f9dd4f7407bd6f7000552），接下来就能构建请求该文档的 URL 了（示例 8-9）。

示例 8-9：从 http://localhost:5984/accounts/3636fa3c716f9dd4f7407bd6f7000552 请求的包含账户信息的 JSON 资源

```
{
  "_id": "3636fa3c716f9dd4f7407bd6f7000552",
  "_rev": "1-8a9527cbfc22e28984dfd3a3e6032635",
  "firstName": "Billy",
  "lastName": "Bob",
  "address": {
      "street": "123 another st",
      "city": "Somewhere",
      "state": "OR",
      "zip": "97501"
  },
  "age": 54,
```

```
  "gender": "male",
  "famous": false
}
```

我们已经了解了如何从 CouchDB 数据库请求数据，接下来看如何向数据库发送数据。

如果我希望在账户数据库中加入一个新的账户，可以通过向 http://localhost:5984/accounts/ 这一 URL post 数据来实现（见示例 8-10 和示例 8-11）。

示例 8-10：通过 `POST` 方法请求 http://localhost:5984/accounts/ 的 HTTP 请求头

```
POST /accounts/ HTTP/1.1
Host: localhost:5984
Content-Type: application/json
Cache-Control: no-cache
```

示例 8-11：通过 `POST` 方法请求 http://localhost:5984/accounts/ 的 HTTP 请求体

```
{
  "firstName": "Janet",
  "lastName": "Jackson",
  "address": {
      "street": "456 Fakey Fake st",
      "city": "Somewhere",
      "state": "CA",
      "zip": "96520"
  },
  "age": 54,
  "gender": "female",
  "famous": true
}
```

> 如果你的浏览器是 Chrome，可以在 Chrome 应用商店中查找并安装 Postman。Postman 提供了一些简单的接口来构建和测试对 API 的 HTTP 请求，并提供了所有的 HTTP 方法（`GET`、`POST`、`PUT`、`DELETE` 等）。同时，Postman 会将请求保存为历史纪录，这样在测试相同的请求时就不必重复创建了。

HTTP 请求成功后，CouchDB API 会将 JSON 格式的响应信息发送给你，其中包含了新创建的文档的标识符（示例 8-12）。

示例 8-12：http://localhost:5984/accounts/ 返回的 JSON 响应

```
{
  "ok": true,
  "id": "3636fa3c716f9dd4f7407bd6f700076c",
  "rev": "1-363f3b4bf90183781d08fe22487f3c90"
}
```

现在可以使用新的唯一标识符来创建 URL，并从账户数据库中请求 JSON 文档（示例 8-13）。

示例 8-13：http://localhost:5984/accounts/3636fa3c716f9dd4f7407bd6f700076c 的响应

```
{
  "_id": "3636fa3c716f9dd4f7407bd6f700076c",
  "_rev": "1-363f3b4bf90183781d08fe22487f3c90",
  "firstName": "Janet",
  "lastName": "Jackson",
  "address": {
      "street": "456 Fakey Fake st",
      "city": "Somewhere",
      "state": "CA",
      "zip": "96520"
  },
  "age": 54,
  "gender": "female",
  "famous": true
}
```

如果希望更新刚刚插入的 JSON 文档，可以通过在 `POST` 请求该资源的 URL 时，将 `"_id"` 和 `"_rev"` 这两个名称 – 值对加入请求体来实现。例如，如果要将 Janet Jackson 的年龄修改为 55，需要将前面示例中的 JSON 文档的年龄字段更新后发送。API 会响应一个说明状态的 JSON 文档，其中会包含更新后的 `"rev"` 名称 – 值对（见示例 8-14）。

示例 8-14：更新 JSON 文档后 http://localhost:5984/accounts/3636fa3c716f9dd4f7407bd6f700076c 的响应

```
{
  "ok": true,
  "id": "3636fa3c716f9dd4f7407bd6f700076c",
  "rev": "3-29ede949ceed9df62bd7caecb095bffe"
}
```

如果希望删除一份文档，需要调用 HTTP 的 `DELETE` 方法，并将 `rev` 标识符作为 query string 的参数传入 URL（示例 8-15）。

示例 8-15：rev 标识符作为 query string 的参数传入 URL

```
http://localhost:5984/accounts/3636fa3c716f9dd4f7407bd6f700076c?
       rev=3-29ede949ceed9df62bd7caecb095bffe
```

如果再次请求 http://localhost:5984/accounts/ 这一 URL 的资源，它会告诉我现在有四份文档（示例 8-16）。

示例 8-16：http://localhost:5984/accounts/ 响应的 JSON 文档，其中 "`doc_count`" 名称 – 值对的值为 4

```
{
    "db_name": "accounts",
    "doc_count": 4,
    "doc_del_count": 0,
    "update_seq": 8,
    "purge_seq": 0,
    "compact_running": false,
    "disk_size": 32869,
    "data_size": 1560,
    "instance_start_time": "1432493477586600",
    "disk_format_version": 6,
    "committed_update_seq": 8
}
```

以上这些例子展示了如何获取和创建存储在 CouchDB 数据库中的 JSON 文档。不过，如果这就是它的全部功能，那么我们很快就有麻烦了。比如，如果我需要一个仅包含名人的姓氏的列表，该怎么办？在 CouchDB 中，可以通过“视图”来实现。见示例 8-17。

视图为我们在 CouchDB 中重组和查询数据提供了方便，它保存在名为设计文档的 JSON 文档中。设计文档具体规定了语言，也可能包含多个视图。

示例 8-17：账户数据库中一个包含名人视图的 CouchDB 设计文档

```
{
  "_id": "_design/lists",
  "_rev": "8-4124de7756277c6a937004a763d6247d",
  "language": "javascript",
  "views": {
      "famous": {
          "map": "function(doc){if(doc.lastName!==null&&doc.famous)
```

```
            {emit
            (doc.lastName,null)}}"
        }
    }
}
```

每个视图都是一个对象，其中可能会包含 `map` 和 `reduce` 函数。示例 8-17 中仅定义了 `map` 函数。`map` 函数会将每个文档看作一个参数，然后调用 `emit()` 函数。这实质上对数据进行了转换（示例 8-18）。

示例 8-18："famous" 视图的 map 函数

```
function(doc) {
    if (doc.lastName !== null && doc.famous) {
        emit(doc.lastName, null)
    }
}
```

示例 8-18 中，函数会检查数据库中的每个 JSON 文档，以确保我们能够得到包含姓氏且 `"famous"` 名称 – 值对的值为 `true` 的数据。`emit()` 的参数是 `key` 和 `value`，它们会在转换后的文档中作为名称 – 值对存在。

接下来就可以通过视图来请求资源了，URL 为 http://localhost:5984/accounts/_design/lists/_view/famous（见示例 8-19）。其结构为 /<db-name>/_design/<design_document_name>/_view/<view_name>/。

示例 8-19：http://localhost:5984/accounts/_design/lists/_view/famous 返回的资源

```
{
    "total_rows": 2,
    "offset": 0,
    "rows": [
        {
            "id": "ddc14efcf71396463f53c0f880001538",
            "key": "Barker",
            "value": null
        },
        {
            "id": "3636fa3c716f9dd4f7407bd6f700076c",
            "key": "Jackson",
            "value": null
        }
    ]
}
```

除了 map 以外，还可以定义 reduce 函数。CouchDB 有三个内置的 reduce 函数：_count、_sum 以及 _stats。在执行完 map 后对数据集中的信息进行统计时，它们非常有用。

例如，如果想知道基于性别的账户数量分别是多少，就可以在设计文档中加入一个新的视图（示例 8-20）。

示例 8-20：在设计文档中加入 "gender_count" 视图

```
{
  "famous": {
      "map": "function(doc){if(doc.lastName!==null&&doc.famous){emit
      (doc.lastName,null)}}"
  },
  "gender_count": {
      "map": "function(doc){if(doc.gender!==null) emit(doc.gender);}",
      "reduce": "_count"
  }
}
```

在 "gender_count" 视图的 map 这一步，emit 函数的参数是 "gender" 名称－值对的值。如果没有 reduce 这一步，它会创建一个由对象构成的数组，其中性别的值保存在 "key" 中（示例 8-21）。

示例 8-21：经过 map 操作后的 "gender_count" 视图

```
{
    "total_rows": 4,
    "offset": 0,
    "rows": [
        {
            "id": "3636fa3c716f9dd4f7407bd6f700076c",
            "key": "female",
            "value": null
        },
        {
            "id": "ddc14efcf71396463f53c0f8800019ea",
            "key": "female",
            "value": null
        },
        {
            "id": "3636fa3c716f9dd4f7407bd6f7000552",
            "key": "male",
            "value": null
        },
        {
```

```
            "id": "ddc14efcf71396463f53c0f880001538",
            "key": "male",
            "value": null
        }
    ]
}
```

通过内置的 "_count" 函数执行过 reduce 这一步之后，得到了将所有记录计数后的结果（示例 8-22）。

示例 8-22：http://localhost:5984/accounts/_design/lists/_view/gender_count 返回的资源

```
{
    "rows": [
        {
            "key": null,
            "value": 4
        }
    ]
}
```

由于想统计的是每个性别的用户的数量，所以需要传入 group 标记来让它去根据 key 的值进行分组。通过加入 URL 的 query string 参数 ?group=true 来实现这一功能。

示例 8-23：http://localhost:5984/accounts/_design/lists/_view/gender_count?group=true 返回的资源

```
{
    "rows": [
        {
            "key": "female",
            "value": 2
        },
        {
            "key": "male",
            "value": 2
        }
    ]
}
```

有了 CouchDB 的 API，就可以为数据库创建任意数量的设计文档和视图了。设计文档中的每一种视图都能包含 map 和 reduce 函数，以转换数据。

然后就可以为存储的 JSON 文档创建自定义的 URL 资源集，来对应处理后的数据集。最终，可以通过 CouchDB API 来为数据创建自己的 API。

8.3 专业术语和概念

本章涵盖了以下专业术语。

- 关系型数据库

  一种将存储的数据用可以辨识的关系进行结构化存储的数据库。

- NoSQL 数据库

  一种不通过存储数据间的关系来存储的数据库。

- CouchDB

  一种面向文档的 NoSQL 数据库存储类型，使用 JSON 文档的形式来存储数据。

我们还讨论了以下重要概念。

- 在关系型数据库中，常常会存在表、列、行以及它们之间的关系，其中会用到主键和外键。
- NoSQL 数据库有许多种，它们拥有与传统的关系型模型不同的数据存储与利用方法。

我们还讨论了 CouchDB，以下是需要记住的重点。

- 它是一种面向文档的 NoSQL 数据库。
- 它存储和管理 JSON 文档。
- 它会在存储与获取数据的同时维护好数据的结构。
- 它使用基于 HTTP 的 API 来获取作为 JSON 文档资源的数据。
- 它使用 JavaScript 作为查询语言，且通过视图的 `map` 和 `reduce` 方法来跨 API 获取数据。

第 9 章

服务端的JSON

当谈及 Web 客户端–服务端关系时，可以将技术按客户端和服务端的区别分类。

- 客户端：HTML、CSS、JavaScript。
- 服务端：PHP、ASP.NET、Node.js、Ruby on Rails、Java、Go，等等。

在 Web 客户端，我们通过 HTTP 向服务端发送资源请求。服务端会响应一份文档，如 HTML 或 JSON。当文档是 JSON 时，必须要用服务端代码来生成它。

同时，服务端可能需要先接收一个 JSON 文档，才能返回一个 JSON 文档。第 8 章中就通过 HTTP 的 `POST` 方法向 CouchDB 的 API 发送过 JSON 文档。当服务端接收到文档后，其代码会对文档进行一系列的解析操作。

在前面的章节，我们了解了 JavaScript 是如何通过幕后的 HTTP 请求来为 Web API 请求 JSON 资源的。JavaScript 是“客户端”的编程语言，让它来扮演这一角色并不难理解。不过，JavaScript 并不是唯一一个可以通过 HTTP 请求获取 JSON 资源的语言。服务端的 Web 框架也可以发起 HTTP 请求。

当通过服务端技术来请求 JSON 时，服务端扮演的其实是客户端的角色。下面列举了一些需要用服务端 Web 框架发送 HTTP 请求的例子。

- 使用 Web API 的数据来渲染一个网站的动态页面。用到的可能是第三方的 Web API，也可能是你自己的私有 Web API，如 CouchDB。
- 通过 Web API 与支付网站通信来集成结账功能的电商网站。
- 通过 Web API 获取到给定地址的实时运费。

本章重点研究 JSON 在服务端扮演的角色。服务端的 Web 框架或脚本语言创建了动态页面。请求资源时，服务端会根据既有的编程逻辑来创建资源。

9.1 序列化、反序列化与请求JSON

一种数据交换格式要想在 Web 领域适用，就必须同时被客户端和服务端支持。如果 JSON 只被客户端支持而不受服务端支持，那么它肯定不会火起来。好消息是，大部分服务端 Web 框架以及脚本语言都较好地支持 JSON。就算没有内置序列化、反序列化 JSON 的功能，也会有库或扩展来支持它。

> 第 6 章中提到过，序列化是将对象转化为文本的过程，反序列化是将文本转化为对象的过程。

下面来看看服务端是如何对 JSON 进行序列化、反序列化以及请求操作的。

9.1.1 ASP.NET

ASP.NET 是由微软开发的服务端 Web 框架。最开始时，该框架仅支持使用 Web Forms——一种 GUI 状态驱动模型。不过现在，它通过扩展允许开发人员使用一些不同的编程模型，如 ASP.NET MVC（一种模型 - 视图 - 控制器架构）和 ASP.NET Web API（一种为 Web 服务构建 HTTP Web API 的架构）。

此外，ASP.NET 使用公共语言运行库（common language runtime，CLR），它允许开发人员使用任何受 CLR 支持的语言来编写代码，如 C#、F#、Visual Basic .NET（VB.NET），等等。这一部分中，所有的示例均使用 C# 编写。

在 ASP.NET 中解析 JSON 不像在 JavaScript 中那么简单。若要解析 JSON，需要使用第三方 ASP.NET 库，并将它带入工程中。写作本书时，实现该功能的最流行的库是 Json.NET，是由 Newtonsoft 开发的开源 JSON 框架。

接下来看看 Json.NET 是如何对 JSON 执行序列化和反序列化操作的。

1. 序列化JSON

有了 ASP.NET 和 Json.NET，可以快速地将 ASP.NET 对象序列化为 JSON。首先，需要一个用于操作的 JSON 对象。本例中（示例 9-1）会创建一个和第 8 章的账户示例结构相同的账户对象。

示例 9-1：用 ASP.NET C# 编写的 CustomerAccount 对象

```
public class CustomerAccount
{
    public string firstName { get; set; }
    public string lastName { get; set; }
    public string phone { get; set; }
    public Address[] addresses { get; set; }
    public bool famous { get; set; }
}

public class Address
{
    public string street { get; set; }
    public string city { get; set; }
    public string state { get; set; }
    public int zip { get; set; }
}
```

既然已经有了表示对象的类，现在就来创建一个新的对象以保存 Bob Barker 的账户信息。在示例的最后一行，我使用 Json.NET 库将对象序列化为 JSON（见示例 9-2 与示例 9-3）。

示例 9-2：用 C# 编写的用于保存 Bob Barker 账户信息的新对象；示例最后将对象序列化为 JSON

```
CustomerAccount bobsAccount = new CustomerAccount();
bobsAccount.firstName = "Bob";
bobsAccount.lastName = "Barker";
bobsAccount.phone = "555-555-5555";

Address[] addresses;
addresses = new Address[2];
```

```
Address bobsAddress1 = new Address();
bobsAddress1.state = "123 fakey st";
bobsAddress1.city = "Somewhere";
bobsAddress1.state = "CA";
bobsAddress1.zip = 96520;

addresses[0] = bobsAddress1;

Address bobsAddress2 = new Address();
bobsAddress2.state = "456 fake dr";
bobsAddress2.city = "Some Place";
bobsAddress2.state = "CA";
bobsAddress2.zip = 96538;

addresses[1] = bobsAddress2;

bobsAccount.addresses = addresses;
bobsAccount.famous = true;

string json = JsonConvert.SerializeObject(bobsAccount);
```

示例 9-3：序列化之后的 ASP.NET `CustomerAccount` 对象

```
{
    "firstName": "Bob",
    "lastName": "Barker",
    "phone": "555-555-5555",
    "addresses": [
        {
            "street": null,
            "city": "Somewhere",
            "state": "CA",
            "zip": 96520
        },
        {
            "street": null,
            "city": "Some Place",
            "state": "CA",
            "zip": 96538
        }
    ],
    "famous": true
}
```

2. 反序列化JSON

仅需一行代码就可以将 JSON 反序列化为 ASP.NET 对象（示例 9-4）。

示例 9-4：前面示例中的 JSON 字符串的反序列化；它被反序列化为了指定的类型，即 CustomerAccount

```
CustomerAccount customerAccount =
    JsonConvert.DeserializeObject<CustomerAccount>(json);
```

Json.NET 允许将 JSON 反序列化为指定类型的对象，如示例中的 CustomerAccount 类型。这样就能保证数据在出入服务端代码时保持自己原有的形态。不过请注意，对象的属性名必须和 JSON 文档中的名称 – 值对的名称相吻合。否则，反序列化操作就会失败。

3. 请求JSON

可以通过使用 ASP.NET 中内置的 System.Net.WebClient 类来创建 HTTP 请求。有了这个类，就能通过调用 DownloadString 方法下载从具体的 URL 请求的资源，资源的类型为字符串（见示例 9-5 与示例 9-6）。

示例 9-5：从本地 CouchDB 地址数据库中下载一份格式为字符串的 JSON 地址文档

```
using(var webClient = new WebClient())
{
    string json = webClient.DownloadString("3636fa3c716f9dd4f7407bd6
    f700076c");
}
```

示例 9-6：http://localhost:5984/accounts/3636fa3c716f9dd4f7407bd6f700076c 的 JSON 资源

```
{
    "_id": "3636fa3c716f9dd4f7407bd6f700076c",
    "_rev": "2-e04207c11104e06b3a8f030f14c35580",
    "address": {
        "street": "456 Fakey Fake st",
        "city": "Somewhere",
        "state": "CA",
        "zip": "96520"
    },
    "age": 54,
    "gender": "female",
    "famous": true,
    "firstName": "Janet",
    "lastName": "Jackson"
}
```

一旦获取了 JSON 资源的字符串，就可以使用 Json.NET 的 `DeserializeObject` 方法将其反序列化为预期的 ASP.NET 对象（示例 9-7）。

示例 9-7：反序列化前面示例中的 JSON 资源的字符串；最后一行的 `fullName` 变量值为“Janet Jackson”

```
CustomerAccount customerAccount;
using (var webClient = new WebClient())
{
    string json = webClient.DownloadString(
        "http://localhost:5984/accounts/3636fa3c716f9dd4f7407bd6f700076c");
    customerAccount = JsonConvert.DeserializeObject<CustomerAccount>(json);
}

string fullName = customerAccount.firstName + " " + customerAccount.
    lastName;
```

在上面这些例子中，JSON 对象的结构和 ASP.NET 对象的结构对应得非常好。只要 ASP.NET 对象的属性和 JSON 的名称 – 值对中的名称能够吻合，在这个 Web 框架中进行 JSON 的序列化和反序列化操作就不会太复杂。

9.1.2 PHP

PHP 是一种用于创建动态 Web 页面的服务端脚本语言。PHP 代码可以直接嵌入 HTML 文档中（见示例 9-8）。

示例 9-8：该 HTML 页面会显示一个内容为 `"Hello, World"` 的标题（`<h1>`）

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Hello, World</title>
    </head>
    <body>
        <h1>
        <?php
           echo "Hello, World";
        ?>
        </h1>
    </body>
</html>
```

此外，PHP 还支持对象数据类型。对象通过类定义（见示例 9-9）。

示例 9-9：表示一只猫的一个 PHP 类

```
class Cat
{
    public $name;
    public $breed;
    public $age;
    public $declawed;
}
```

类的实例化意味着对象的创建。之后该对象便可在编程逻辑中使用（示例 9-10）。

示例 9-10：将类实例化并设置属性。这意味着对象被创建了。最后一行会输出 "Fluffy Boo"

```
$cat = new Cat();
$cat->name = "Fluffy Boo";
$cat->breed = "Maine Coon";
$cat->age = 2.5;
$cat->declawed = false;

echo $cat->name;
```

PHP 对 JSON 的序列化和反序列化操作也有内置的支持。这类操作在 PHP 中称为 JSON 的编码与解码。对某样东西进行编码的意思是将它转为一种编码格式（即不可读的形式）。解码的意思则是将它转为一种可读的格式。对于 PHP 来说，JSON 是一种编码格式。因此，序列化 JSON 时应调用 `json_encode` 函数，反序列化 JSON 时应调用 `json_decode` 函数。

1. 序列化JSON

在 PHP 中，可以通过内置的 JSON 支持快速地将 PHP 对象序列化。这一部分中，我们将创建一个保存地址信息的 PHP 对象，并将其序列化为 JSON。

示例 9-11 中，先创建一个保存账户信息的类，然后创建一个该类的新实例来保存 Bob Barker 的账户信息。`Account` 对象在这一步被创建。最后，代码的最后一行调用了内置的 `json_encode` 函数将 `Account` 对象序列化（结果展示在示例 9-12 中）。

示例 9-11：创建并序列化一个账户

```
<?php
class Account {
    public $firstName;
```

```
    public $lastName;
    public $phone;
    public $gender;
    public $addresses;
    public $famous;
}

class Address {
    public $street;
    public $city;
    public $state;
    public $zip;
}

$address1 = new Address();
$address1->street = "123 fakey st";
$address1->city = "Somewhere";
$address1->state = "CA";
$address1->zip = 96027;

$address2 = new Address();
$address2->street = "456 fake dr";
$address2->city = "Some Place";
$address2->state = "CA";
$address2->zip = 96345;

$account = new Account();
$account->firstName = "Bob";
$account->lastName = "Barker";
$account->gender = "male";
$account->phone = "555-555-5555";
$account->famous = true;
$account->addresses = array ($address1, $address2);

$json = json_encode($account);

?>
```

示例 9-12：json_encode($account) 返回的结果

```
{
    "firstName": "Bob",
    "lastName": "Barker",
    "phone": "555-555-5555",
    "gender": "male",
    "addresses": [
        {
            "street": "123 fakey st",
            "city": "Somewhere",
```

```
                "state": "CA",
                "zip": 96027
            },
            {
                "street": "456 fake dr",
                "city": "Some Place",
                "state": "CA",
                "zip": 96345
            }
        ],
        "famous": true
    }
```

2. 反序列化JSON

使用 PHP 内置的 json_encode 函数序列化 JSON。相对地，使用 json_decode 函数来反序列化 JSON。不过，内置的方法并不支持将 JSON 反序列化为某种具体的 PHP 对象类型，如 Account 类。所以，需要采取一些措施将数据重塑为对应的 PHP 对象。

首先给 Account 对象添加一个新函数，使它可以通过 JSON 字符串设置自己的属性。

示例 9-13 中，loadFromJSON 函数接受一个 JSON 字符串作为参数，它会调用内置的 json_decode 函数反序列化一个一般的 PHP 对象，然后在 foreach 循环中遍历该对象的属性，并赋值给 Account 对象中名称相同的属性。

示例 9-13：向 Account 对象添加一个函数

```
class Account {
    public $firstName;
    public $lastName;
    public $phone;
    public $gender;
    public $addresses;
    public $famous;

    public function loadFromJSON($json)
    {
        $object = json_decode($json);
        foreach ($object AS $name => $value)
        {
            $this->{$name} = $value;
        }
    }
}
```

接下来创建一个新的 `Account` 对象，并调用新的 `loadFromJSON` 函数（示例 9-14）。

示例 9-14：调用 `loadFromJSON` 函数来将账户的 JSON 反序列化为 `Account` 对象；最后一行会输出 "Bob Barker"

```
$json = json_encode($account);

$deserializedAccount = new Account();
$deserializedAccount->loadFromJSON($json);

echo $deserializedAccount->firstName . " " . $deserializedAccount->
lastName;
```

3. 请求JSON

使用 PHP 内置的 `file_get_contents` 函数来创建 HTTP 请求。该函数会将资源以字符串的形式返回。然后就可以将字符串反序列化为 PHP 对象（见示例 9-15 与示例 9-16）。

示例 9-15：本地 CouchDB API 在 http://localhost:5984/accounts/ddc14efcf-71396463f53c0f8800019ea 响应的资源

```
{
    "_id": "ddc14efcf71396463f53c0f8800019ea",
    "_rev": "6-69fd853972074668f99b88a86aa6a083",
    "address": {
        "street": "123 fakey ln",
        "city": "Some Place",
        "state": "CA",
        "zip": "96037"
    },
    "gender": "female",
    "famous": false,
    "age": 28,
    "firstName": "Mary",
    "lastName": "Thomas"
}
```

示例 9-16：调用内置的 `file_get_contents` 函数请求上例中的 JSON 资源。然后创建一个新的 `Account` 对象，并调用 `loadFromJSON` 进行反序列化操作。最后一行将输出 "Mary Thomas"

```
$url = "http://localhost:5984/accounts/3636fa3c716f9dd4f7407bd6f700076c";
$json = file_get_contents($url);
```

```
$deserializedAccount = new Account();
$deserializedAccount->loadFromJSON($json);

echo $deserializedAccount->firstName . " " . $deserializedAccount->
lastName;
```

9.2 发送JSON HTTP请求的其他方式

从 PHP 和 ASP.NET 的示例来看，虽然 JSON 基于 JavaScript 的对象字面量表示法，但这种表示法也可以很方便地转换为其他编程语言中的对象。服务端编程语言多种多样，且大多数都支持对象和 JSON 数据类型。这使得数据可以用 JSON 这种格式以它原本的结构在系统中进出。

此外，服务端对 JSON 提供了强有力的支持。大多数服务端语言（或框架）要么提供了内置的 JSON 序列化 / 反序列化支持，要么有提供响应的库或模块。服务端的有力支持是 JSON 在数据交换格式中脱颖而出的重要原因。

接下来看看服务端上一些请求 JSON 的小例子。在每个小例子中，都能看到使用 HTTP 请求 JSON 资源，将资源解析为对象，以及 JSON 文档中的值的渲染。

下面所有示例使用的 JSON 文档都来源于 OpenWeatherMap API。渲染的值都基于 http://api.openweathermap.org/data/2.5/weather?q=London,uk 这一 JSON 文档的子集（见示例 9-17）。

示例 9-17：OpenWeatherMap API 中 JSON 资源的坐标对象，其值代表伦敦的经纬度

```
{
    "coord": {
        "lon": -0.13,
        "lat": 51.51
    }
}
```

9.2.1 Ruby on Rails

Ruby on Rails 是一个服务端的 Web 应用框架。它使用 Ruby 语言编写且基于模型 – 视图 – 控制器（MVC）架构。

Ruby 中的 gem 是一个用来安装程序或库的包。为在 Ruby on Rails 中对 JSON 进行序列化和反序列化操作，我们需要 JSON ruby gem。一旦装好了 JSON ruby gem，解析 JSON 就变得很简单了，只需执行 `JSON.parse()`（见示例 9-18）。

示例 9-18 中，向 OpenWeatherMap API 发送请求，并将 JSON 反序列化为 Ruby 对象。借助代码的最后一行，我们在页面中渲染出了数据中坐标（`coord`）对象中的经度值（`lon`）。

示例 9-18：向 OpenWeatherMap API 发送请求

```
require 'net/http'
require 'json'

url = URI.parse('http://api.openweathermap.org/data/2.5/
weather?q=London,uk')
request = Net::HTTP::Get.new(url.to_s)
response = Net::HTTP.start(url.host, url.port) {|http|
http.request(request)
}

weatherData = JSON.parse(response.body)

render text: weatherData["coord"]["lon"]
```

9.2.2 Node.js

Node.js 是服务端的 JavaScript（脱离了浏览器），它基于谷歌的开源 JavaScript 引擎 V8。有了 Node.js，就可以使用 JavaScript 编写服务端应用。

前面的章节探讨过 JSON 与客户端的 JavaScript，并通过 `JSON.parse()` 轻松地将 JSON 反序列化为 JavaScript 对象。因为 Node.js 也是 JavaScript，所以方法都是一样的。

不过，在 Node.js 中不再使用 `XMLHttpRequest` 对象，因为该对象是仅在互联网浏览器中使用的 JavaScript 对象。在 Node.js 中，通过更简单的 `get()` 函数来请求 JSON（以及其他类型的资源）。

示例 9-19 中，向 OpenWeatherMap API 发送请求，并将得到的 JSON 反序列化为 JavaScript 对象。然后通过 `console.log()` 输出坐标（`coord`）对象中

的经度值（lon）。

示例 9-19：向 API 发送 HTTP 请求并将其反序列化为 JavaScript 对象

```
var http = require('http');
http.get({
    host: 'api.openweathermap.org',
    path: '/data/2.5/weather?q=London,uk'
}, function(response) {
    var body = '';
    response.on('data', function(data) {
        body += data;
    });
    response.on('end', function() {
        var weatherData = JSON.parse(body);
        console.log(weatherData.coord.lon);
    });
});
```

9.2.3 Java

Java 是一种面向对象的编程语言。它可以以 Java 小应用的形式运行在 Web 浏览器中，也可以通过 Java 运行环境（Java runtime environment，JRE）单独在机器上运行。

Java 中的许多库都提供了对 JSON 的支持。在这一部分中，我将使用下面的库来通过 URL 获取 JSON，并将其序列化为 Java 对象。

- Apache Commons IO
- JSON in Java

示例 9-20 中，从 OpenWeatherMap API 中以字符串的形式获取 JSON 资源。然后通过实例化 `JSONObject` 将 JSON 字符串反序列化。最后使用 `System.out.println()` 输出坐标（`coord`）对象中的经度值（`lon`）。

示例 9-20：通过 `JSONObject` 反序列化 JSON 字符串

```
import java.io.IOException;
import java.net.URL;

import org.apache.commons.io.IOUtils;
import org.json.JSONException;
import org.json.JSONObject;
```

```
public class HelloWorldApp {

    public static void main(String[] args) throws IOException,
    JSONException {
        String url = "http://api.openweathermap.org/data/2.5/
        weather?q=London,uk";
        String json = IOUtils.toString(new URL(url));
        JSONObject weatherData = new JSONObject(json);
        JSONObject coordinates = weatherData.getJSONObject("coord");
        System.out.println(coordinates.get("lon"));
    }

}
```

9.3 专业术语和概念

本章涵盖了以下专业术语。

- ASP.NET
  微软开发的服务端 Web 框架。

- PHP
  用于创建动态 Web 页面的服务端脚本语言。

- Ruby on Rails
  使用 Ruby 编写的服务端 Web 应用框架。

- Node.js
  基于谷歌 V8 引擎的服务端 JavaScript。

- Java
  一种面向对象编程语言。

我们还讨论了以下重要概念。

- 在服务端，可以通过将 JSON 反序列化为对象而运用在编程逻辑中，也可以将对象序列化为 JSON 格式。
- JSON 同时被客户端和服务端较好地支持，使得它在 Web 领域从诸多数据交换格式中脱颖而出。

第 10 章

总结

纵观全网，JSON 在各种地方传递着数据。作为一种数据交换格式，它是成功的，本书已经充分论证了这一点。不过，如果深入了解它的应用，会发现它还会作为配置文件静止地待在某个地方。

没错，JSON 还会作为静止的配置文件大放异彩，不过，本书没有过多深入这个主题。本书主要篇幅都在讨论数据交换，也就是数据的传递。

JSON 既是数据的容器，也是运输工具，它的用途并没有局限于某一个方面。我们知道它在 NoSQL 中是存储数据的载体。而且，JSON 也在逐渐成为项目中有用的工具。为了进一步说明，让我们最后来看看 JSON 的另一用途：作为配置文件放在某一个地方。

10.1 作为配置文件的JSON

第 9 章主要探究了两种服务端 Web 技术是如何解析 JSON 的。那一章中提到了 JSON 被大多数服务端 Web 开发语言（或框架）所支持。由于具备这种广泛的支持、服务端解析的便利性以及良好的可读性，JSON 常用来存储配置信息。

软件中经常会有配置文件或设置文件，它让我们可以不必重新编译就能修改设置。配置文件的格式有很多，有 INI 和 XML 等。

让我们来比较一下使用 INI、XML、JSON 三种格式存储同样设置的区别（见示例 10-1、示例 10-2 和示例 10-3）。

示例 10-1：使用 INI 格式的某游戏配置文件（settings.ini）

```
[general]
playIntro=false
mouseSensitivity=0.54

[display]
complexTextures=true
brightness=4.2
widgetsPerFrame=326
mode=windowed

[sound]
volume=1
effects=0.68
```

示例 10-2：使用 XML 格式的某游戏配置文件（settings.xml）

```
<?xml version="1.0" encoding="UTF-8" ?>
<settings>
    <general>
        <playIntro>false</playIntro>
        <mouseSensitivity>0.54</mouseSensitivity>
    </general>
    <display>
        <complexTextures>true</complexTextures>
        <brightness>4.2</brightness>
        <widgetsPerFrame>326</widgetsPerFrame>
    </display>
    <sound>
        <volume>1</volume>
        <effects>0.68</effects>
    </sound>
</settings>
```

示例 10-3：使用 JSON 格式的某游戏配置文件（settings.json）

```
{
    "general": {
        "playIntro": false,
        "mouseSensitivity": 0.54
    },
    "display": {
        "complexTextures": true,
        "brightness": 4.2,
        "widgetsPerFrame": 326,
```

```
            "mode": "windowed"
        },
        "sound": {
            "volume": 1,
            "effects": 0.68
        }
    }
```

这三种格式都有着良好的可读性。如果希望修改声音设置，就能从很多文件中快速定位出这个文件，并且对数值进行修改。这就是它们作为配置文件的一大优势。

每种格式都有优点和缺点。INI 格式没有 XML 那些大于号和小于号以及 JSON 的花括号，所以有更好的可读性。不过 INI 格式不能很好地表示更为复杂的信息，如嵌套信息或复杂的列表。XML 能够包含更为复杂的数据，但是它不像 JSON 一样具有数据类型。

除了这些数据格式本身具有的优缺点外，是否能够很方便地被编程语言 / 框架解析也是一个很重要的考量因素。如果 JSON 解析器已经在你的应用中深度使用了，那么 JSON 可能是你配置文件的最佳选择。

第 9 章简单讲解了服务端 Node.js 的 JSON 操作。现实中一个使用 JSON 作为配置文件的极佳例子就是 Node.js 默认的 JavaScript 包管理器：npm。当然，它也被 AngularJS 和 jQuery 等其他框架使用。

npm 包管理器使用 JSON 作为配置文件，文件名称为 package.json。该文件包含了每个包的具体信息，如名称、版本、作者、贡献者、依赖、脚本以及许可（见示例 10-4）。

示例 10-4：npm 的 package.json 示例文件

```
{
  "name": "bobatron",
  "version": "1.0.0",
  "description": "The extraordinary library of Bob.",
  "main": "bob.js",
  "scripts": {
    "prepublish": "coffee -o lib/ -c src/bob.coffee"
  },
  "repository": {
    "type": "git",
    "url": "git://github.com/bobatron/bob.git"
```

```
    },
    "keywords": [
      "bob",
      "tron"
    ],
    "author": "Bob Barker <bob.barker@fakemail.com> (http://bob.com/)",
    "license": "BSD-3-Clause",
    "bugs": {
      "url": "https://github.com/bobatron/issues"
    }
  }
```

尽管 JSON 是作为配置文件放在一个地方的，但从某种意义上讲，它还是在扮演着数据交换格式的角色。对于存放在某处的配置文件来说，交换仍在人和计算机之间进行着，而数据则在其中起着沟通作用。

10.2 结语

无论是作为服务器上的配置文件，还是作为通过 URL 请求的资源，JSON 始终都在履行作为数据交换格式的职责。本书中，我们一步步探索了这些职责（或是角色）。让我们最后概括一下 JSON 是如何在我们的生活中发挥作用的。

在服务端，对象可以被序列化为 JSON 格式的文本，并且通过反序列化变回对象。服务端代码可以请求 JSON。在第 9 章中，我们了解了如何用 ASP.NET 和 PHP 去实现这一功能。

此外，JSON 也可以被看作一种文本格式，用作一种面向文档存储类型的数据库的文档。第 8 章介绍了使用此方法的 CouchDB。该数据库同时提供了可用于 HTTP Web API 的接口。

HTTP Web API 是一个对诸如 HTML 或 JSON 文档等资源进行请求和响应的系统。这些文档使用 URL 经由 HTTP 请求。第 6 章就讲解了 OpenWeatherMap API 是如何利用这一点以 JSON 资源的形式提供天气数据的。

JavaScript `XmlHttpRequest` 可以通过 URL 请求 JSON 资源。为了在 JavaScript 代码中使用 JSON，要先将它反序列化为 JSON 对象。JavaScript

内置的 `JSON.parse()` 函数能够快速而有效地实现这一功能。

从 JavaScript 对象到 JSON，再到服务端的对象，能够看到数据跑了一圈。每一天，数据都在全世界各种系统中进进出出，它们的载体就是数据交换格式。

我们再次从高空俯瞰，可以发现 JSON 并不是唯一一种数据交换格式。有些数据以逗号分隔值（CSV）为载体，还有些是 Excel 表格。它们都是表格化的。还有些数据以 XML 格式存在，这种格式支持数据的嵌套。

数据有多种格式与形式。有多种不同的系统在不停地推送和接收着数据。在考虑使用什么数据交换格式时，数据的形式和交换数据的系统都应该被考虑到。虽说本书青睐于 JSON，并对它进行了全方位的介绍，但要记住，JSON 不总是最佳选择。

例如，假设两个不同的系统需要交换库存数据，它们都使用表格的形式来存储这些数据，那就有必要将表格数据转换成对象的形式，序列化成 JSON，再到另一个系统上转成对象，最后再转换成表格形式吗？答案是否定的，因为这样就等于专门为了使用 JSON 而南辕北辙了。应该为表格形式的数据选择一种更合适的数据交换格式，比如 CSV 或 tab delimited。

由于多数系统都使用对象来为数据建模，所以 JSON 作为数据交换格式非常流行。数据常被保存在关系型数据库中。尽管关系型数据库使用表格，但实际上还是可以通过数据的关系来构成可以视为对象的实体。每个实体，如地址，都有一些“字段”，这些字段实际上就是名称 – 值对。

在互联网浏览器这类通过 JavaScript 来支持面向对象编程的系统中，JSON 是较为理想的数据交换格式。在使用 JavaScript 与 Web API 交流或是创建 AJAX 交互时，JSON 都有极佳的表现。

对于那些诸如面向对象的服务端 Web 框架之类的系统，JSON 依旧是理想的数据交换格式。有了 JSON，数据就可以以对象字面量的形式在系统中进出，并保持它原有的结构。

就像本章 10.1 节“作为配置文件的 JSON”所提到的例子那样，选择格式前应先考虑优缺点。用对的工具，去做对的工作。

作者简介

Lindsay Bassett 对技术的写作与教学都有着很高的热情。她的在线技术课程和书籍都展现了一致的“直切要害”的风格，非常适合学生以及业务繁忙的 IT 人士。她善于提炼复杂主题的核心枝干，并且善于将枯燥的技术书籍写得十分有趣。

封面介绍

本书封面上的动物是斑海豹（Phoca vitulina），是一种常见于北美洲与欧洲海岸线的温带和寒温带水域的海洋哺乳动物。它们是“真正的海豹”，这意味着它们身体前侧有短鳍状肢，有肥硕的香肠状身体，没有外耳。

每一只斑海豹都有着独特的斑点形式和皮肤颜色：肤色可能是银色、灰色、棕色、蓝灰色或黄褐色，上面点缀着或明或暗的斑点，斑点的明暗取决于主肤色。它们的平均体长为 6 英尺，体重从 120 磅到 370 磅不等；雌性的体型比雄性略小一些。

尽管斑海豹在上岸时姿态笨拙（摇来摇去），但其圆滑的骨架和强有力的尾部鳍状肢使得它们的水下活动十分敏捷强大。由于它们有洄游行为，因此食物也会随季节变化，但主要食物还是各种各样的鱼类、乌贼、甲壳类和软体动物。它们可以下潜到 1500 英尺的地方，在水下活动长达 40 分钟，但比较常见的还是持续 3~7 分钟的浅层潜水。

斑海豹一生中有一半时间是在岸上度过——它们上岸主要是为了休息和恢复体温，也会进行群集活动和繁殖行为。斑海豹通常都会有固定的休憩场所和繁殖场所，但如果有人类经常造访的话，它们就会遗弃这些场所。因此，是否开发海豹活动频繁的海岸需要我们慎重考虑。

斑海豹的幼崽刚一出生就会游泳，只需几天，它们就可以潜水两分钟了。四至六周后它们会断奶，但由于母海豹的奶水中含有大量脂肪，因此在哺乳期间它们的体重几乎会增加一倍。

O’Reilly 图书封面上的许多动物都是濒危物种；所有这些动物都对我们的世界有着重要意义。如果你希望了解你力所能及的事，请访问 animals.oreilly.com。

封面图片来自 Lydekker 的 *Royal Natural History*。

延 展 阅 读



JavaScript 编程入门书

通过学习 Web 开发了解计算机科学的基本思想和原理

书号：978-7-115-41816-6

定价：89.00 元



构建先行，设计干净、可测试、结构良好的 JavaScript 应用

书号：978-7-115-40210-3

定价：59.00 元

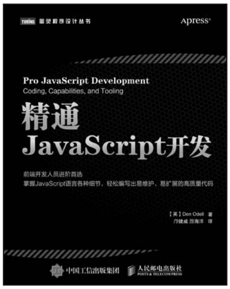


业界最先进的动画库 Velocity.js 作者作品

揭秘开发人员如何用动画轻松提升用户体验

书号：978-7-115-41012-2

定价：39.00 元



前端开发人员进阶首选

掌握 JavaScript 语言各种细节，轻松编写出易维护、易扩展的高质量代码

书号：978-7-115-40255-4

定价：79.00 元

延 展 阅 读

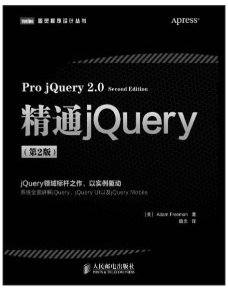


前后端程序员必备指南

全面讲解最新的 C/S 应用开发范式

书号：978-7-115-39730-0

定价：59.00 元



jQuery 领域标杆之作

以实例驱动，系统全面讲解 jQuery、jQuery UI 以及 jQuery Mobile

书号：978-7-115-36653-5

定价：149.00 元



CSS 一姐 Lea Verou 作品

近年来最重要的 CSS 技术书

挖掘了很多有用的 CSS 技术点

书号：978-7-115-41694-0

定价：99.00 元



Web 页面视觉盛宴，展示近 700 个最出色的网页

设计师进阶必读，总结一流网页应遵守的设计原则

分门别类介绍，囊括各种网页风格

书号：978-7-115-40211-0

定价：49.00 元

O'REILLY®

JSON必知必会

JSON（JavaScript对象表示法）是一种流行的数据交换格式，从Web API和服务端编程语言到NoSQL数据库和客户端框架，都有JSON的身影。在不同平台间传递数据方面，JSON已成为XML强有力的替代者。

本书将帮助忙碌的IT从业者快速学习JSON，并且深入理解如何将其用在自己的项目中。书中对JSON的语法、数据类型、格式和安全问题等进行了全面的解读，并且展示了JSON在实际生活中的诸多用途。只要你略有编程经验并且对HTML和JavaScript有基本的了解，就可以轻松阅读本书。

- 了解为什么JSON语法中使用名称－值对表示数据
- 掌握JSON中的对象、字符串、数字和数组等数据类型
- 探讨如何解决常见的安全问题
- 学习使用JSON模式来验证数据格式是否正确
- 审视浏览器、Web API和JSON之间的关系
- 理解服务端如何请求和创建数据
- 探索如何在jQuery及其他客户端框架中使用JSON
- 分析为何CouchDB NoSQL数据库使用JSON存储数据

Lindsay Bassett是作家、教育工作者和Web开发者，对技术教学与技术写作都有极大的热情。她的在线技术课程和图书都简明扼要，非常适合繁忙的IT人士和学生。

“整本书都在讲JSON？是的，这本书涵盖JSON的方方面面，甚至包含你并不知道但需要知道的JSON知识。这是一份信息丰富而全面的优秀资源。”

——Shelley Powers
Web开发者，
*JavaScript Cookbook*与
*HTML5 Media*作者

WEB PROGRAMMING

封面设计：Karen Montgomery　张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议　计算机／软件开发

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

ISBN 978-7-115-42207-1

9 787115 422071 >

ISBN 978-7-115-42207-1
定价：35.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：
ebook@turingbook.com。

在这可以找到我们：

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks